Security Analysis of the Unbalanced Oil and Vinegar Signature Scheme

Ostap Cherkashin

Department of Mathematics
The University of Auckland

Supervisor: Professor Steven D. Galbraith

A thesis submitted in partial fulfilment of the requirements for the degree of
MSc in Mathematics, The University of Auckland, 2022.

Abstract

Due to advancements in quantum computer algorithms, the cryptographic community is exploring alternatives to traditional hardness assumptions, such as the discrete logarithm problem. One such assumption is the Multivariate Quadratic (MQ) problem, which is used by a number of cryptographic primitives, including the Unbalanced Oil and Vinegar (UOV) signature scheme. Apart from MQ, UOV uses another, less understood, hardness assumption which we call the polynomial equivalence problem. The reliance on an open problem and simplicity of the scheme make UOV an excellent subject for cryptographic research.

This work studies the UOV construction and methods of solving multivariate systems of nonlinear equations that underlie the security of the scheme. The thesis is split into three parts: key space of UOV, polynomial system solving, and the polynomial equivalence problem. The contributions include a complete classification of sustaining transformations, a study of Gröbner bases complexity estimation, and implementations of algorithms. Security assessment of UOV is presented in the final chapter.

Contents

Chapter 1

Introduction

Unbalanced Oil and Vinegar (UOV) is a digital signature scheme that belongs to a class of cryptographic constructions collectively known as the Multivariate Public Key Cryptosystems (MPKC). Among the main benefits of UOV are performance, short signature size, and presumed resistance to quantum algorithms.

In 2016, the National Institute of Standards and Technology initiated a collaborative process to identify, study, and standardize a set of algorithms for Post-Quantum Cryptography (NIST PQC) [Nat22]. Since MPKC is presumed to be resilient to attacks by quantum computers there were a number of proposals from this class of cryptosystems. UOV has not been submitted as a candidate signature algorithm, but there were several related constructions, such as Rainbow [DS05] and LUOV [BPSV19]. These algorithms aimed to minimize the key lengths at the cost of introducing additional complexity. As of this writing, both Rainbow and LUOV have been broken [Beu22, DDVY21]. On the other hand, UOV remains unbroken for certain parameter choices for over two decades. Perhaps we will see UOV in the future submissions to NIST PQC.

As with other cryptographic constructions, the security of UOV is based on presumed hardness of certain problems as well as a general assumption that $\mathsf{P} \neq \mathsf{NP}$. In particular, there are two hardness assumptions underlying the security of UOV:

1. The problem of finding solutions to multivariate systems of quadratic equations. This problem has been demonstrated to be NP-complete for random systems of polynomial equations [GJ79].

2. The Polynomial Equivalence Problem (PEP). Informally, given a system of multivariate quadratic equations the goal is to find a decomposition of the system into a linear map and a quadratic form of a certain structure. There are no known reductions of this problem

to an NP-complete problem.

The motivation of this work is to study how the hardness assumptions above as well as the specifics of the UOV construction correspond to claimed security levels. In particular, we explore the following questions:

1. Does the structure of a UOV central map help to solve public keys directly? We noted above that a randomly generated polynomial system cannot be solved efficiently using currently known techniques. However, we do not know if the polynomial systems obtained from central maps enjoy the same property.

2. Is PEP a sound security assumption? Despite being at the forefront of the security claims of UOV this problem appears to be not well understood. At the same time, attacks on certain parameter selections of UOV appear to be instantiations of PEP.

Organization of the Thesis

The thesis is split into five chapters with Chapters 2, 3, and 4 forming the main body of work.

Chapter 2 defines the Unbalanced Oil and Vinegar signature scheme and studies the structure of the UOV key space. The main contribution is the complete classification of sustaining transformations presented by Theorem 2.3.7.

Chapter 3 studies methods of solving multivariate polynomial systems. This is a large topic, so we restrict our attention to developing complexity bounds on computing Gröbner bases and studying two additional algorithms that are of interest to MPKC. One of the contributions of Chapter 3 is the development of Gröbner basis complexity estimates from the ground up. Since complexity of solving polynomial systems is still an active area of research, we bridge the gap between the introductory material and the current research. Additionally, we present an implementation of the Thomae-Wolf algorithm and provide a compact proof of existence of the respective solutions. This algorithm suggests that the number of equations in UOV is the defining parameter of the complexity.

Chapter 4 introduces the polynomial equivalence problem for UOV. We study three key results in this field: Kipnis-Shamir attack, reconciliation attack, and intersection attack. The main contributions of this chapter are the PEP framework and the implementation of the reconciliation attack. The framework enables reductions of PEP to other problems. We develop a reduction to the oil space O_x in Theorem 4.1.3 and show how the existing attacks fit into this category. Further ideas of reductions include the stabilizer group $\mathrm{stab}(p)$ and permutation matrices briefly discussed in Chapter 5 under future work.

Acknowledgements

I am deeply grateful for all the support and guidance provided by my research supervisor Professor Steven D. Galbraith. Working on this thesis has been one of the most exciting journeys in my life. I would also like to thank Jeroen Schillewaert whose encouraging words motivated me to pursue more challenging but rewarding paths.

Chapter 2

Unbalanced Oil and Vinegar

Unbalanced Oil and Vinegar (UOV) is a digital signature scheme introduced by Aviad Kipnis, Jacques Patarin, and Louis Goubin [KPG99]. We will provide the corresponding definitions and introduce a suitable notation.

Definition 2.0.1 (Digital Signature)**.** Let k be a field of positive characteristic, $m, n \in \mathbb{N}$. A digital signature is a tuple $(\mathsf{K_p}, \mathsf{K_s}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{KeyGen})$, where

(i) $\mathsf{K_p}$ is a set of public keys
(ii) $\mathsf{K_s}$ is a set of secret keys
(iii) $\mathsf{Sign} : \mathsf{K_s} \times k^m \to k^n$ is a map that produces digital signatures
(iv) $\mathsf{Verify} : \mathsf{K_p} \times k^m \times k^n \to \{0, 1\}$ is a map that verifies digital signatures
(v) KeyGen is a probabilistic algorithm that produces key pairs $(p, s) \in \mathsf{K_p} \times \mathsf{K_s}$ such that

$$\mathsf{Verify}(p, \mu, \mathsf{Sign}(s, \mu)) = 1$$

for all messages $\mu \in k^n$. This is known as the correctness property.

A particular specification of the key space $\mathsf{K_p} \times \mathsf{K_s}$ and the corresponding maps Sign, Verify, KeyGen is called a *digital signature scheme*. The corresponding components of UOV are based on multivariate polynomials.

Definition 2.0.2 (Multivariate Polynomial)**.** A multivariate polynomial f in indeterminates $x_1, \ldots, x_n$ over a field k is a linear combination of monomials $x^\alpha = x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ of the form

$$f = \sum_{\alpha \in A} c_\alpha x^\alpha,$$

where $c_\alpha \in k$ and A is a finite subset of $\mathbb{Z}_{\geq 0}^n = \{(\alpha_1, \ldots, \alpha_n) \mid \alpha_i \in \mathbb{Z}_{\geq 0}, 1 \leq n \leq n\}$. The set of all multivariate polynomials in $x_1, \ldots, x_n$ over k is denoted by $k[x_1, \ldots, x_n]$.

A polynomial $f \in k[x_1, \ldots, x_n]$ defines a map $f : k^n \to k$ using polynomial evaluation $(a_1, \ldots, a_n) \mapsto f(a_1, \ldots, a_n)$ for all $(a_1, \ldots, a_n) \in k^n$. Similarly, an m-tuple of polynomials $g = (g_1, \ldots, g_m)$ from $k[x_1, \ldots, x_n]^m$ defines a map $g : k^n \to k^m$ via

$$(a_1, \ldots, a_n) \mapsto (g_1(a_1, \ldots, a_n), \ldots, g_m(a_1, \ldots, a_n)).$$

Finding a pre-image $a \in k^n$ such that $g(a) = b$ for some $b \in k^m$ is known to be NP-complete for a randomly chosen $g \in k[x_1, \ldots, x_n]^m$. This also applies to quadratic polynomials over finite fields [GJ79, DPS20], which motivates the construction of UOV and other multivariate public key cryptosystems.

Informally, the idea of UOV is to use polynomials of a certain structure, which is easily invertible, then hide this structure via a linear change of variables to produce a public key. This special form of polynomials is captured by the following definition.

Definition 2.0.3 (Central Map)**.** Let n, m be positive integers with $n > m$. We say that an ordered sequence of polynomials $f = (f_1, \ldots, f_m)$ in $k[x_1, \ldots, x_n]^m$ is a central map of Unbalanced Oil and Vinegar if

$$f_\ell = \sum_{i=1}^{n-m} \sum_{j=i}^{n} c_{i,j} x_i x_j \tag{2.1}$$

for all $1 \leq \ell \leq m$ with $c_{i,j} \in k$. The set of all polynomials of the form (2.1) is denoted by S.

Observe that the components f_ℓ of a central map are homogeneous of degree two, so we can write them as upper triangular matrices in $k^{n \times n}$ of quadratic forms

$$F_\ell = \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & \cdots & \cdots & c_{1,n} \\ 0 & c_{2,2} & \cdots & \cdots & \cdots & c_{2,n} \\ \vdots & \ddots & \ddots & & & \vdots \\ 0 & \cdots & 0 & c_{n-m,n-m} & \cdots & c_{n-m,n} \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ \vdots & & & & & \vdots \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 \end{bmatrix}.$$

Lemma 2.0.4 (Pre-image of a Central Map)**.** *Let $n > m$, $f = (f_1, \ldots, f_m) \in S^m$, and $\mu = (\mu_1, \ldots, \mu_m) \in k^m$. Assume that the system of equations $f(\sigma_1, \ldots, \sigma_{n-m}, x_{n-m+1}, \ldots, x_n)$ in $k[x_{n-m+1}, \ldots, x_n]^m$ behaves as a random system for randomly chosen values $\sigma_1, \ldots, \sigma_{n-m}$ in k. Then, there is a $\sigma = (\sigma_1, \ldots, \sigma_n) \in k^n$ such that $f(\sigma) = \mu$ with probability strictly greater than $1 - \frac{1}{q-1}$, where q is the order of k.*

Proof. Choose $(\sigma_1, \ldots, \sigma_{n-m}) \in k^{n-m}$ uniformly at random and evaluate

$$f_\ell(\sigma_1, \ldots, \sigma_{n-m}, x_{n-m+1}, \ldots, x_n) = \sum_{i=n-m+1}^{n} d_i x_i + d$$

for all $1 \leq \ell \leq m$, where $d_i, d \in k$. This produces a system of m linear equations $f'_1, \ldots, f'_m$ in m variables $x_{n-m+1}, \ldots, x_n$. Next, we compose a linear system

$$\begin{aligned} f'_1 &= \mu_1 \\ &\vdots \\ f'_m &= \mu_m \end{aligned}$$

and attempt to solve it for $x_{n-m+1}, \ldots, x_n$. If a solution exists, we obtain the pre-image σ, where the last m elements come from the solution $\sigma_{n-m+\ell} = x_{n-m+\ell}$ for $\ell = 1, \ldots, m$. The probability that a solution exists is determined by the probability that the coefficient matrix of f'_ℓ is invertible, which occurs with a probability bounded below by $1 - \frac{1}{q-1}$ by Corollary 2.1.3. □

If a pre-image has not been found from the first attempt, it is possible to select a different value for $(\sigma_1, \ldots, \sigma_{n-m}) \in k^{n-m}$ and retry. This is precisely what is done in practice. Due to nonzero probability of finding a pre-image $\sigma \in k^n$ when the linear system of equations $f'_1, \ldots, f'_m$ behaves as a random system, we assume that central maps are invertible.

Definition 2.0.5 (Unbalanced Oil and Vinegar)**.** Let k be a field of positive characteristic and suppose $n > m$. The components of the Unbalanced Oil and Vinegar signature scheme are defined as follows

(i) The public key space is $\mathsf{K_p} = k[x_1, \ldots, x_n]^m$
(ii) The secret key space is $\mathsf{K_s} = S^m \times \mathrm{GL}(n, k)$
(iii) Sign operation maps a secret key $(f, A) \in \mathsf{K_s}$ and a message $\mu \in k^m$ into a signature $\sigma = (A^{-1} \circ f^{-1})(\mu)$ in k^n. This is a probabilistic algorithm as per Lemma 2.0.4.
(iv) Verify operation maps a public key $p \in \mathsf{K_p}$, a message $\mu \in k^m$, and a signature σ to 1 if $p(\sigma) = \mu$. Otherwise, Verify returns 0.
(v) KeyGen chooses a secret key $s = (f, A)$ from $\mathsf{K_s}$ uniformly at random and composes a public key $p = f \circ A$ in $\mathsf{K_p}$.

Note that the correctness property specified in Definition 2.0.1.(v) holds by Lemma 2.0.4 since

$$\sigma = (A^{-1} \circ f^{-1})(\mu) \implies p(\sigma) = (f \circ A) \circ (A^{-1} \circ f^{-1})(\mu) = \mu,$$

for all $\mu \in k^m$. Therefore, UOV forms a digital signature scheme.

Remark 1 (Simplified UOV). The original definition of UOV [KPG99] is slightly different from the one presented above. The secret transformation A is defined to be an invertible affine map $A(x) = Bx + c$ for some $B \in \mathrm{GL}(n, k)$ and $c \in k^n$. The central map f is defined to have linear and constant terms, which makes the public key p also contain linear and constant terms. Definition 2.0.3 is known as the "simplified UOV" and is commonly used in the analysis of the signature scheme [KS98, KPG99, DPS20, Beu21]. An instance of UOV based on the original definition can be transformed into the simplified form by homogenizing the public key p with respect to a new variable x_0. We will restrict our attention to the simplified UOV.

Remark 2 (Alternative Key Generation). Stanislav Bulygin, Albrecht Petzoldt, and Johannes Buchmann proposed an alternative key generation algorithm for UOV [BPB10]. Instead of choosing a random central map $f \in S^m$ and a random transformation $A \in \mathrm{GL}(n, k)$, it is possible to choose a subset of coefficients of a public key $p \in R^m$ and a linear map A, then compute a central map f using the relationships between the coefficients of p and the elements of A. Once f is determined, the complete public key p is computed using $f \circ A$.

The name of the signature scheme is assumed to originate from a salad dressing that uses oil and vinegar as ingredients. Oil and vinegar are mixed together, but they do not combine. Extending this analogy to definitions above, the input variables to a central map $f \in S^m$ are split into two parts

$$\underbrace{a_1, \ldots\ldots, a_{n-m}}_{\text{Vinegar}}, \underbrace{a_{n-m+1}, \ldots, a_n}_{\text{Oil}}.$$

These variables are "mixed" by a secret linear transformation A but it is possible to recover the variables using A^{-1}.

Example 2.0.6 (Sign and Verify Operations)**.**
We will generate a small UOV key pair, sign a message, and verify the signature. This example was produced by the program `uov.sage` available in Appendix A.1.

Let k be a field of order 31, $n = 5$, $m = 2$. Even for such small parameter values the polynomials contain too many terms to be readable. Therefore, we will use matrices of the respective quadratic forms. A random linear transformation along with a random central map are given by

$$A = \begin{bmatrix} 0 & 23 & 27 & 1 & 23 \\ 19 & 14 & 15 & 22 & 4 \\ 7 & 5 & 13 & 7 & 12 \\ 5 & 24 & 4 & 2 & 10 \\ 30 & 3 & 15 & 10 & 13 \end{bmatrix} \quad F_1 = \begin{bmatrix} 15 & 13 & 6 & 9 & 25 \\ 0 & 21 & 28 & 8 & 18 \\ 0 & 0 & 24 & 24 & 11 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad F_2 = \begin{bmatrix} 26 & 1 & 5 & 13 & 5 \\ 0 & 27 & 0 & 21 & 16 \\ 0 & 0 & 28 & 16 & 9 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

and the respective public key is

$$P_1 = \begin{bmatrix} 22 & 30 & 15 & 13 & 15 \\ 0 & 26 & 23 & 4 & 21 \\ 0 & 0 & 14 & 25 & 26 \\ 0 & 0 & 0 & 12 & 20 \\ 0 & 0 & 0 & 0 & 10 \end{bmatrix} \quad P_2 = \begin{bmatrix} 8 & 29 & 15 & 27 & 13 \\ 0 & 3 & 4 & 8 & 24 \\ 0 & 0 & 17 & 25 & 13 \\ 0 & 0 & 0 & 26 & 4 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Our goal is to sign the message $\mu = (7, 14)$. We choose $(\sigma'_1, \sigma'_2, \sigma'_3) = (18, 7, 29)$ at random and substitute into f_1 and f_2 to obtain linear equations

$$f'_1 = 15x_4 - 4x_5 + 9$$
$$f'_2 = 8x_4 - 2x_5 + 9.$$

The corresponding matrix

$$B = \begin{bmatrix} 15 & -4 \\ 8 & -2 \end{bmatrix}$$

is invertible, so we can immediately solve the system of equations given by $f'_1 = 7$ and $f'_2 = 14$. This yields solutions $\sigma'_4 = x_4 = 12$ and $\sigma'_5 = x_5 = 30$. The complete pre-image of the central map f is thus $\sigma' = (18, 7, 29, 12, 30)$. Next, we compute a pre-image for the public key via $\sigma = A^{-1}(\sigma')^\intercal$ and obtain the UOV signature $\sigma = (26, 16, 17, 19, 26)$. To verify the signature, we compute $\sigma^\intercal P_1 \sigma = 7$ and $\sigma^\intercal P_2 \sigma = 14$, which matches the message μ.

We will assume the following notation for the rest of the document. Some chapters may explicitly assign a different meaning to a symbol, but, if the meaning is not specified, it should be assumed to be as follows.

Symbol	Meaning
n	Number of variables $x_1, \ldots, x_n$
m	Number of polynomials in a system $p_1, \ldots, p_m$
R	All homogeneous polynomials of degree two in $k[x_1, \ldots, x_n]$
S	All central maps in R
p	Public key $p = (p_1, \ldots, p_m) \in R^m$ or a single component $p \in R$
f	Central map $f = (f_1, \ldots, f_m) \in S^m$ or a single component $f \in S$.

2.1 Basic Counting of the Key Space

The set of all possible key combinations of UOV can be defined as a Cartesian product of central maps with the invertible linear transformations

$$K = S^m \times \mathrm{GL}(n, k).$$

Note, there is no need to account for public keys separately since they are merely compositions of elements of this set.

We will denote by $M(x, n)$ the set of all monomials in $k[x_1, \ldots, x_n]$. That is,

$$M(x, n) = \{x^\alpha = x_1^{\alpha_1} \cdots x_n^{\alpha_n} \mid \alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{Z}_{\geq 0}^n\}.$$

It is convenient to restrict monomials to a particular degree, so we additionally introduce the following notation

$$M_d(x, n) = \{x^\alpha \in M(x, n) : \deg(x^\alpha) = d\},$$
$$M_{\leq d}(x, n) = \{x^\alpha \in M(x, n) : \deg(x^\alpha) \leq d\}.$$

When the number and names of the variables $x_1, \ldots, x_n$ is clear from the context we will simply write M, M_d, $M_{\leq d}$ instead of $M(x, n)$, $M_d(x, n)$, $M_{\leq d}(x, n)$, respectively.

Lemma 2.1.1. *The number of distinct monomials in n variables of degrees d and $\leq d$ is given by*

$$|M_d(x, n)| = \binom{n+d-1}{d} \quad \textit{and} \quad |M_{\leq d}(x, n)| = \binom{n+d}{d}.$$

Proof. To determine $|M_d|$ we need to count the number of ways to split a d-element set into n subsets, which is the same as choosing an arrangement of $n-1$ separators in a set of d objects and $n-1$ separators. That is,

$$|M_d| = \frac{(d + (n-1))!}{d!(n-1)!} = \binom{n+d-1}{d}.$$

The second claim follows immediately by

$$|M_{\leq d}| = \sum_{i=0}^{d} \binom{n+i-1}{i} = \binom{n+d}{d}.$$

□

Let q denote the order of field k. For each distinct monomial, there are q choices for the value of the coefficient in k. This yields a formula for the number of homogeneous polynomials of degree two

$$|R| = q^{\frac{1}{2}n(n+1)}.$$

The number of possible public keys is then bounded above by $|R|^m$, because we have m polynomials in each public key.

Input variables x_i are split into two subsets called Oil and Vinegar variables respectively:

$$\underbrace{x_1, \ldots\ldots, x_{n-m}}_{\text{Vinegar}} \underbrace{x_{n-m+1}, \ldots, x_n}_{\text{Oil}}.$$

While any subset of m variables can be denoted as Oil variables, the common arrangements are either the last m variables, as illustrated above, or the first m variables. We will use the arrangement depicted above throughout this document.

Remark 3 (Permutations of Variables). By Kerckhoffs's principle, a particular implementation of UOV under investigation must be considered public, so the arrangement of indices does not increase the key space. On the contrary, if the implementation uses indices $1, \ldots, m$ for Oil variables, but a private key is found whose central map is linear in $x_{n-m+1}, \ldots, x_n$, the key is invertible using the same means as the standard signing procedure[1].

Lemma 2.1.2. *The cardinalities of the main constituents of the UOV key space are as follows:*

(i) $|\mathrm{GL}(n,k)| = \prod_{i=0}^{n-1}(q^n - q^i)$
(ii) $|R| = q^{\frac{1}{2}n(n+1)}$
(iii) $|S| = q^{\frac{1}{2}(n-m)(n+m+1)}$
(iv) $|S^{\complement}| = q^{\frac{1}{2}n(n+1)}\left(1 - q^{-\frac{1}{2}m(m+1)}\right)$, *where the complement* $S^{\complement}$ *is taken with respect to* R.

Proof. (i) Since the linear transformations in $\mathrm{GL}(n,k)$ are invertible, they are onto and so must be defined by n linearly independent vectors. There are $q^n - 1$ choices for the first vector v_1 because we must exclude 0. The i-th vector v_i is chosen from q^n vectors that are not in the span of the preceding $i-1$ vectors defined by $W = \mathrm{span}\{\alpha_1 v_1 + \cdots + \alpha_{i-1}v_{i-1} \mid \alpha_j \in k\}$. Now $|W| = q^{i-1}$ because there are q choices for each coefficient α_j. Consequently, there are $q^n - q^{i-1}$ choices for v_i and the general result follows.

(ii) Follows by the monomial counting argument above.

(iii) Write a central map $f \in S$ as a polynomial with distinct monomials and coefficients $\alpha_{ij} \in k$

$$f = \sum_{i=1}^{n-m}\sum_{j=i}^{n}\alpha_{ij}x_ix_j = \sum_{i=1}^{n-m}\sum_{j=i}^{n-m}\alpha_{ij}x_ix_j + \sum_{i=1}^{n-m}\sum_{j=n-m+1}^{n}\alpha_{ij}x_ix_j.$$

On the right hand side, the first sum has $(n-m)(n-m+1)/2$ terms and the second sum has $(n-m)m$ terms, so there are

$$\frac{1}{2}(n-m)(n-m+1) + (n-m)m = \frac{1}{2}(n-m)(n+m+1)$$

[1] Accounting for permutations of variables is further discussed in Chapter 5.

terms in total.

(iv) Since S and $S^{\mathbb{C}}$ are disjoint in R, we have $|S^{\mathbb{C}}| = |R| - |S|$. Observe that the power of q in the expression for $|S|$ can be written as

$$\begin{aligned}\frac{1}{2}(n-m)(n+m+1) &= \frac{1}{2}n(n+m+1) - \frac{1}{2}nm - \frac{1}{2}m(m+1) \\ &= \frac{1}{2}n(n+1) - \frac{1}{2}m(m+1).\end{aligned} \tag{2.2}$$

Then the result follows by factoring $|R|$ from the difference below:

$$\left|S^{\mathbb{C}}\right| = |R| - |S| = q^{\frac{1}{2}n(n+1)} - q^{\frac{1}{2}n(n+1)-\frac{1}{2}m(m+1)}.$$

□

An immediate corollary that is worth highlighting is that random square matrices are invertible with a fairly high probability.

Corollary 2.1.3. *Let A be a matrix in $k^{n\times n}$ chosen uniformly at random. Then the probability that A is invertible is strictly greater than $1 - \frac{1}{q-1}$, where q is the order of the field k.*

Proof. The probability that a randomly chosen matrix is invertible is determined by

$$\frac{|\mathrm{GL}(n,k)|}{|k^{n\times n}|} = \frac{\prod_{i=0}^{n-1}(q^n - q^i)}{q^{n^2}} = \prod_{i=0}^{n-1}(1-q^{i-n}) = 1 - \sum_{i=1}^{n}\left(\frac{1}{q^i}\prod_{j=i+1}^{n}\left(1-\frac{1}{q^j}\right)\right)$$

where the first equality is by Lemma 2.1.2.(i). The products $\prod_{j=i+1}^{n}(1-q^{-j}) < 1$, so

$$\frac{|\mathrm{GL}(n,k)|}{|k^{n\times n}|} > 1 - \sum_{i=1}^{n} q^{-i} > 1 - \sum_{i=1}^{\infty} q^{-i} = 1 - \frac{1}{q-1}$$

using geometric series and the fact that $q \geq 2$ in $\mathbb{Z}$. □

The probability that a randomly chosen polynomial in R is a central map is now easy to compute using Lemma 2.1.2 and Equation (2.2). It is defined by the following ratio

$$\frac{|S|}{|R|} = q^{-\frac{1}{2}m(m+1)}.$$

Note that the probability is independent of whether we sample polynomials $g \in R$ or compositions $g \circ A$ for some linear transformation A, provided the sampling is uniform. If we let $A \in \mathrm{GL}(n,k)$ and define $\psi_A : R \to R$ by $\psi_A(g) = g \circ A$, then the map is a bijection since A is invertible. Therefore, the ratio of $|S|$ to $|R|$ is preserved by ψ_A for all $A \in \mathrm{GL}(n,k)$.

2.2 Equivalent Keys

The size of the UOV key space $K = S^m \times \mathrm{GL}(n,k)$ is given by $|K| = |S|^m|\mathrm{GL}(n,k)|$, which can be easily expanded into a formula in q, n, m using Lemma 2.1.2. The subject of this section is to study whether the elements in K are unique and to further breakdown the key space into useful subsets.

Definition 2.2.1 (Equivalent Keys)**.** We say that the private key $(f, A) \in K$ is equivalent to a private key $(f', A') \in K$ if $f \circ A = f' \circ A'$ as polynomials in R.

The definition above yields an equivalence relation on K

$$(f, A) \sim (f', A') \iff f \circ A = f' \circ A' \text{ in } R.$$

Reflexivity, symmetry, and transitivity hold since the map composition $f \circ A$ is well-defined and polynomials are equal in R if and only if all the coefficients are equal. Therefore, the key space is partitioned into equivalence classes

$$[p] = \{(f, A) \in K \mid p = f \circ A\},$$

where we identified a public key $p = f' \circ A'$ as a representative element of the class.

We are interested in estimating the size of the equivalence classes, which can be achieved using the sustaining transformation framework of Christopher Wolf and Bart Preneel [WP05].

Definition 2.2.2 (Sustaining Transformation)**.** A linear map $A \in \mathrm{GL}(n,k)$ is called a sustaining transformation of Unbalanced Oil and Vinegar if $f \circ A \in S$ for all $f \in S$.

A trivial example of a sustaining transformation is the identity element of $\mathrm{GL}(n,k)$. Let $p = f \circ T$ be a public key. If $A \in \mathrm{GL}(n,k)$ is a sustaining transformation, then

$$p = f \circ (AA^{-1}T) = (f \circ A) \circ (A^{-1}T) = f' \circ T',$$

where $f' \in S$ and $T' \in \mathrm{GL}(n,k)$. That is, a sustaining transformation produces an equivalent key. Therefore, we can determine the number of equivalent keys by estimating the number of sustaining transformations. An invaluable tool in this respect is that a matrix of quadratic form of a central map has a particular structure.

Lemma 2.2.3. *Let $f \in S$ and denote by $F \in k^{n\times n}$ the matrix of quadratic form of f. Then F has a zero submatrix of dimensions $m \times m$ corresponding to the indices of the oil variables. That is, for our choice of the indices,*

$$f \in S \iff F = \begin{bmatrix} * & * \\ * & 0_{m\times m} \end{bmatrix} \in k^{n\times n}.$$

Proof. ($\Leftarrow$) Write a vector $x = (x_1, \ldots, x_n) \in k[x_1, \ldots, x_n]^n$ and a matrix $F \in k^{n\times n}$ of the central map as

$$x = \begin{bmatrix} x_v \\ x_o \end{bmatrix} \text{ and } F = \begin{bmatrix} F_1 & F_2 \\ F_3 & 0_{m\times m} \end{bmatrix},$$

where $x_v = (x_1, \ldots, x_{n-m})$ and $x_o = (x_{n-m+1}, \ldots, x_n)$ denote the components of x corresponding to vinegar and oil variables respectively. Then

$$f = x^\intercal F x = \begin{bmatrix} x_v^\intercal & x_o^\intercal \end{bmatrix} \begin{bmatrix} F_1 & F_2 \\ F_3 & 0 \end{bmatrix} \begin{bmatrix} x_v \\ x_o \end{bmatrix} = x_v^\intercal F_1 x_v + x_v^\intercal F_2 x_o + x_o^\intercal F_3 x_v.$$

Since oil and vinegar variables do not overlap in x, the resulting polynomial f is linear in the oil variables x_o and so $f \in S$.

($\Rightarrow$) Given a polynomial $f \in S$ that is linear in x_o we can write it as

$$f = x_v^\intercal G_1 x_v + x_v^\intercal G_2 x_o = \begin{bmatrix} x_v^\intercal & x_o^\intercal \end{bmatrix} \begin{bmatrix} G_1 & G_2 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_v \\ x_o \end{bmatrix} = x^\intercal G x$$

for some matrices $G_1 \in k^{(n-m)\times(n-m)}$ and $G_2 \in k^{(n-m)\times m}$. □

The first estimate of the number of equivalent keys was presented by Wolf and Preneel [WP05, Theorem 3] based on the following observation

$$A^\intercal F = \begin{bmatrix} A_1^\intercal & 0 \\ 0 & A_4^\intercal \end{bmatrix} \begin{bmatrix} F_1 & F_2 \\ F_3 & 0 \end{bmatrix} = \begin{bmatrix} A_1^\intercal F_1 & A_1^\intercal F_2 \\ A_4^\intercal F_3 & 0 \end{bmatrix},$$
$$FA = \begin{bmatrix} F_1 & F_2 \\ F_3 & 0 \end{bmatrix} \begin{bmatrix} A_1 & 0 \\ 0 & A_4 \end{bmatrix} = \begin{bmatrix} F_1 A_1 & F_2 A_4 \\ F_3 A_1 & 0 \end{bmatrix}.$$

Therefore, $A^\intercal F A$ has the form of a central map, which makes matrices of the form A sustaining transformations. The count of sustaining transformations of the form A follows immediately by Lemma 2.1.2

$$\prod_{i=0}^{m-1} (q^m - q^i) \prod_{i=0}^{n-m-1} (q^{n-m} - q^i),$$

since $|A_1| = |\mathrm{GL}(m, k)|$ and $|A_4| = |\mathrm{GL}(n-m, k)|$. A better estimate can be obtained using Theorem 2.3.7, which we will prove in the next section.

Theorem 2.2.4. *For a fixed public key $p \in R^m$ the number of equivalent keys is bounded below by*

$$\prod_{i=0}^{m-1} (q^n - q^{n-m+i}) \prod_{i=0}^{n-m-1} (q^{n-m} - q^i).$$

Proof. By Theorem 2.3.7, all matrices A of the following form yield sustaining transformations

$$A = \begin{bmatrix} *_{(n-m)\times(n-m)} & 0_{(n-m)\times m} \\ *_{m\times(n-m)} & *_{m\times m} \end{bmatrix}.$$

Moving row by row from the top down and constructing linearly independent vectors as in the proof of Lemma 2.1.2, we have $|\mathrm{GL}(n-m,k)|$ choices for the top left square submatrix. Once this submatrix is fixed, the number of choices for the first vector that is not in the span of the preceding $n-m$ vectors is $q^n - q^{n-m}$. Then for the $(i+1)$-th vector in the bottom m rows we have $q^n - q^{n-m+i}$ choices and the formula above follows. □

We can now estimate the probability of making a poor choice of the secret linear transformation during the key generation procedure.

Lemma 2.2.5. *The probability that a randomly chosen central map $f \in S$ remains linear in the Oil variables after the linear change of variables $f \circ A$ for a randomly chosen $A \in \mathrm{GL}(n,k)$ is bounded below by*

$$\prod_{i=0}^{n-m-1} \frac{q^{n-m} - q^i}{q^n - q^i}.$$

Proof. Assuming uniform distribution, this probability is equal to the number of sustaining transformations to the number of elements in $\mathrm{GL}(n,k)$. By Theorem 2.2.4 we have a lower bound on the former value. Re-indexing the variables in the leftmost product of Theorem 2.2.4 to start with $i = n-m$ and combining with Lemma 2.1.2 produces

$$\frac{\prod_{i=n-m}^{n-1}(q^n - q^i)\prod_{i=0}^{n-m-1}(q^{n-m} - q^i)}{\prod_{i=0}^{n-1}(q^n - q^i)},$$

and the result follows. □

2.3 Further Insights Using Group Actions

The sections above were mainly concerned with $\mathrm{GL}(n,k)$ and S^m components of the UOV key space $K = S^m \times \mathrm{GL}(n,k)$ independently of each other. Group actions provide means to bridge this gap by considering how the elements of $\mathrm{GL}(n,k)$ act on the polynomials in R^m.

Recall that, given a set X and a group G, a map $\varphi : X \times G \to X$ is called a right group action of G on X if φ satisfies the following properties:

(i) Identity: $\varphi(x,e) = x$ for all $x \in X$ and the identity element $e \in G$,
(ii) Compatibility: $\varphi(\varphi(x,g),h) = \varphi(x,gh)$.

For our purposes we want to define φ via map composition $h \circ A$ of homogeneous polynomials h of degree two and an invertible linear transformation A. Let $\varphi : R^m \times \mathrm{GL}(n,k) \to R^m$ be defined by $\varphi(h, A) = h \circ A$. Expanding the components of $h = (h_1, \ldots, h_m)$, the image of φ is as follows

$$\varphi((h_1, \ldots, h_m), A) = (h_1 \circ A, \ldots, h_m \circ A)$$

The map is well-defined since composition $h_i \circ A$ is well-defined. Identity holds because $\varphi(h, 1) = h$. Compatibility holds because $\varphi(\varphi(h, A), B) = h \circ A \circ B = \varphi(h, AB)$ due to the fact that A, B are linear. Therefore, φ is a right group action.

It is customary to work with left group actions as opposed to right group actions, so we want to write $\varphi(h, A) = A \cdot h$ to be consistent with other sources. We can identify a left group action with the right group action by considering the opposite group of $\mathrm{GL}(n,k)$ defined by the same underlying set and the change of operation from $(\cdot)$ to $(\diamond)$ as follows $A \cdot B = B \diamond A$.

Lemma 2.3.1. *The map $\varphi : \mathrm{GL}(n,k) \times R^m \to R^m$ defined by $\varphi(A, f) = f \circ A$ for all $f \in R^m$ and all $A \in \mathrm{GL}(n,k)$ is a group action. We denote the image $\varphi(A, f)$ by $A \cdot f$.*

Interpreting the elements of R as matrices of quadratic forms, the action of A on $h \in R$ can be written as $A \cdot h = A^\intercal H A$, where H is the matrix corresponding to h. Therefore the three operations are identical

$$A \cdot h = h \circ A = A^\intercal H A.$$

Definition 2.3.2. The orbit of $h \in R^m$ with respect to the general linear group is the set

$$\mathrm{orb}(h) = \{A \cdot h \in R^m \mid A \in \mathrm{GL}(n,k)\}.$$

The stabilizer group of an element $h \in R^m$ is a subset of $\mathrm{GL}(n,k)$ that fixes h. That is,

$$\mathrm{stab}(h) = \bigcap_{i=1}^{m} \{A \in \mathrm{GL}(n,k) \mid A \cdot h_i = h_i\}.$$

Even though we defined $\mathrm{stab}(h)$ as a set, it was called a group with some hindsight. The name is well-deserved for any characteristic of k, because the set is closed under products and contains inverse elements.

Orbits partition R^m into disjoint sets using the following equivalence relation

$$h \sim h' \iff h' = A \cdot h,$$

for some $A \in \mathrm{GL}(n,k)$. Of particular interest are the orbits produced by the UOV central maps $f \in S^m$. If $p = A \cdot f \in R^m$ is a public key, then $p \in \mathrm{orb}(f)$. Furthermore, if (A', f') is an equivalent private key, then $p = A \cdot f = A' \cdot f'$ implies $f = A^{-1}A' \cdot f'$. In other words, f and f' must also share the orbit.

Lemma 2.3.3. *If $p = A \cdot f \in R^m$ is a public key and (A', f') is an equivalent private key, then*

$$\mathrm{orb}(p) = \mathrm{orb}(f) = \mathrm{orb}(f').$$

The orbit of a central map f also contains all public keys that can be derived from f, so it is of little use in cryptanalysis. However, there could be more than one orbit in R^m.

Lemma 2.3.4. *If $p = T \cdot f \in R^m$ is a public key, then*

$$|\mathrm{stab}(p)| = |\mathrm{stab}(f)| = \frac{|\mathrm{GL}(n,k)|}{|\mathrm{orb}(f)|}.$$

Proof. Let $\psi : \mathrm{stab}(f) \to \mathrm{stab}(p)$ be defined by $\psi(A) = TAT^{-1}$ for all $A \in \mathrm{stab}(f)$. The map ψ is an inner automorphism of $\mathrm{GL}(n,k)$, so $|\mathrm{stab}(p)| = |\mathrm{stab}(f)|$ will follow as soon as we show that the image of ψ is in $\mathrm{stab}(p)$.

If $A \in \mathrm{stab}(f)$ then $A \cdot f = f$. On the other hand, $f = T^{-1} \cdot p$ and $p = TA \cdot f$ by compatibility of the group action φ. Combining these results produces $p = TAT^{-1} \cdot p$, so $TAT^{-1} = \psi(A)$ is in $\mathrm{stab}(p)$.

The last equality follows by the Orbit-Stabilizer Theorem [Hum96, Theorem 10.16]. □

Recall that a subset S of R is said to be invariant under $I \subseteq \mathrm{GL}(n,k)$ if $I \cdot S \subseteq S$, where the product $I \cdot S$ is defined by

$$I \cdot S = \{A \cdot f \mid A \in I \text{ and } f \in S\}.$$

Note that, unlike a stabilizer group $\mathrm{stab}(f) \subseteq \mathrm{GL}(n,k)$, the invariant $I \subseteq \mathrm{GL}(n,k)$ does not require the elements of S to be fixed. That is, a weaker condition of $A \cdot h \in S$ suffices instead of $A \cdot h = h$. With this in mind, we make the following definition.

Definition 2.3.5. The set I is a subset of invertible linear transformations that preserve all central maps. That is, I is the set of all sustaining transformations defined by

$$I = \{A \in \mathrm{GL}(n,k) \mid A \cdot f \in S \text{ for all } f \in S\}.$$

The inclusion $I \cdot S \subseteq S$ follows from the definition, but the converse also holds because $1 \in \mathrm{GL}(n,k)$ preserves all elements of S, so $I \cdot S = S$.

Lemma 2.3.6. *The set I is a subgroup*[2] *of $\mathrm{GL}(n,k)$.*

[2]However, I is not a normal subgroup of $\mathrm{GL}(n,k)$. See the discussion immediately after the proof of Theorem 2.3.7.

Proof. Since k is finite so is $\mathrm{GL}(n,k)$, so we only need to show that I is closed with respect to multiplication. Fix $f \in S$ and $A, B \in I$, then $AB \cdot f = A \cdot f' = f''$ for some $f', f'' \in S$ be the definition of I. □

Theorem 2.3.7. *Each sustaining transformation of Unbalanced Oil and Vinegar has a zero submatrix in the top-right. That is,*

$$A \in I \iff A = \begin{bmatrix} * & 0_{(n-m)\times m} \\ * & * \end{bmatrix}.$$

Proof. ($\Leftarrow$) Let $f \in S$ and denote by F the matrix of quadratic form f. By Lemma 2.2.3, F has a zero submatrix corresponding to the indices of the oil variables. Let A be of the form above. Then

$$FA = \begin{bmatrix} F_1 & F_2 \\ F_3 & 0 \end{bmatrix} \begin{bmatrix} A_1 & 0 \\ A_3 & A_4 \end{bmatrix} = \begin{bmatrix} F_1A_1 + F_2A_3 & F_2A_4 \\ F_3A_1 & 0 \end{bmatrix},$$
$$A^\intercal F = \begin{bmatrix} A_1^\intercal & A_3^\intercal \\ 0 & A_4^\intercal \end{bmatrix} \begin{bmatrix} F_1 & F_2 \\ F_3 & 0 \end{bmatrix} = \begin{bmatrix} A_1^\intercal F_1 + A_3^\intercal F_3 & A_1^\intercal F_2 \\ A_4^\intercal F_3 & 0 \end{bmatrix}.$$

Therefore, $A^\intercal FA$ must have a zero submatrix in the bottom right. Applying Lemma 2.2.3 in the opposite direction implies that $f \circ A \in S$. In other words, A is a sustaining transformation[3].

($\Rightarrow$) Suppose there is a matrix

$$A = \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix} \in I,$$

such that the submatrix $A_2 \neq 0$. Consider the matrix F of an arbitrary quadratic form $f \in S$. By Lemma 2.2.3, F has a zero submatrix in the bottom right. Then,

$$A \cdot f = A^\intercal FA = \begin{bmatrix} * & * \\ * & 0 \end{bmatrix},$$

where the last equality follows by Lemma 2.2.3 in the opposite direction. Multiplying $A^\intercal FA$ produces

$$\begin{bmatrix} A_1^\intercal & A_3^\intercal \\ A_2^\intercal & A_4^\intercal \end{bmatrix} \begin{bmatrix} F_1 & F_2 \\ F_3 & 0 \end{bmatrix} \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix} = \begin{bmatrix} * & * \\ * & A_2^\intercal F_1 A_2 + A_2^\intercal F_2 A_4 + A_4^\intercal F_3 A_2 \end{bmatrix}.$$

In particular, the following holds for all $F \in k^{n\times n}$, since we have already taken into account that F must have the structure of the central map

$$A_2^\intercal F_1 A_2 + A_2^\intercal F_2 A_4 + A_4^\intercal F_3 A_2 = 0. \tag{2.3}$$

[3]Direction ($\Leftarrow$) of Theorem 2.3.7 is an extension of [DPS20, Lemma 5.10] that additionally covers $A^\intercal F$ from an overview of the reconciliation attack by Ding et al. It appears that the authors were not aware that direction ($\Rightarrow$) also holds, which provides a complete classification of sustaining transformations of UOV.

Denote the elements of the $(n-m) \times n$ submatrix A_2 by a_{ij} and the elements of the $(n-m) \times (n-m)$ submatrix F_1 by b_{ij}. Since $A_2 \neq 0$ there must be some nonzero element a_{rs}. However, the equation (2.3) must hold for all $F \in k^{n \times n}$, so we set $F_2 = F_3 = 0$ and we set all elements of F_1 to zero except for $b_{rr} = 1$. But then the product $A_2^{\mathsf{T}} F_1 A_2 = (c_{ij})$ must be equal to zero, yet the element $c_{ss} = a_{rs}^2 \neq 0$. This is a contradiction. □

An interesting application of Theorem 2.3.7 is to show that I is not a normal subgroup of $\mathrm{GL}(n,k)$. Suppose I is normal then $B^{-1}AB$ must be in I for all $A \in I$ and all $B \in \mathrm{GL}(n,k)$. But $B^{-1}AB$ is a change of basis and we can easily find a basis that moves some nonzero elements into the top-right submatrix of A.

Lemma 2.3.8. *The subgroup $I \subseteq \mathrm{GL}(n,k)$ preserves public keys:*

$$I \cdot S^{\mathfrak{C}} = S^{\mathfrak{C}}.$$

Proof. Since $1 \in I$, we only need to show that $I \cdot S^{\mathfrak{C}} \subseteq S^{\mathfrak{C}}$. Let $A \in I$ and let P be a matrix of quadratic form $p \in S^{\mathfrak{C}}$. By Theorem 2.3.7, the top-right submatrix of A is zero, so

$$A \cdot p = \begin{bmatrix} A_1^{\mathsf{T}} & A_3^{\mathsf{T}} \\ 0 & A_4^{\mathsf{T}} \end{bmatrix} \begin{bmatrix} P_1 & P_2 \\ P_3 & P_4 \end{bmatrix} \begin{bmatrix} A_1 & 0 \\ A_3 & A_4 \end{bmatrix} = \begin{bmatrix} * & * \\ * & A_4^{\mathsf{T}} P_4 A_4 \end{bmatrix}.$$

Suppose $A \cdot p \notin S^{\mathfrak{C}}$, then $A \cdot p \in S$, which by Lemma 2.2.3 implies $A_4^{\mathsf{T}} P_4 A_4 = 0$. Now $P_4 \neq 0$ since $p \in S^{\mathfrak{C}}$, so A_4 must either be zero or singular. However, both cases contradict that A is invertible. □

2.4 Key Space Statistics

Using the results from the discussion above we will now compute different statistics about the UOV key space. This should illustrate how large the components of the key space are relative to each other. The statistics is computed using a small selection of published UOV parameter sets summarized in Table 2.1. We will augment the selection with a few small parameter values.

Table 2.2 provides counting statistics about the public keys and central maps. Table 2.3 shows the effect of equivalent keys on the key space and compares the results obtained in this thesis with prior work on sustaining transformations [WP05]. Table 2.4 shows the likelihood of making a poor choices during the key generation process.

We conclude this section with Figure 2.1, which illustrates how all components of the UOV key space uncovered in this section fit together. This summarizes our current knowledge about the structure of the key space.

m	n	q	$n = cm$	Comments
44	103	2^8	$n \approx 2.34m$	An optimistic parameter choice due to Czypek et al. [CHT12]. It is no longer considered to provide 128 bits of security because of the intersection attack (Section 4.4).
48	144	2^8	$n = 3m$	A set of parameters for 128-bit security level published by Ding et al. [DPS20, Table 5.1].
64	192	2	$n = 3m$	A set of parameters from the sustaining transformation framework of Wolf and Preneel [WP05]. We include this to simplify the comparison.

Table 2.1: A selection of published UOV parameters.

m	n	q	$\log_2 \lvert R^m \rvert$	$\log_2 \lvert S^m \rvert$	$\log_2 \lvert \mathrm{GL}(n,k) \rvert$	$\log_2 \lvert K \rvert$
2	5	2	30	24	23	47
2	5	31	148	118	123	242
44	103	256	1 885 312	1 536 832	84 871	1 621 703
48	144	256	4 008 960	3 557 376	165 887	3 723 263
64	192	2	1 185 792	1 052 672	36 862	1 089 534

Table 2.2: Counts of different components using Lemma 2.1.2.

m	n	q	$\log_2 \lvert K \rvert$	K' red.	$\log_2 \lvert K' \rvert$	K'' red.	$\log_2 \lvert K'' \rvert$
2	5	2	47	9	37	15	31
2	5	31	242	64	178	94	148
44	103	256	1 621 703	43 335	1 578 368	64 103	1 557 600
48	144	256	3 723 263	92 159	3 631 104	129 023	3 594 240
64	192	2	1 089 534	20 476	1 069 057	28 668	1 060 865

Table 2.3: Key space reductions via sustaining transformations. Column $|K|$ indicates the size of the key space via direct computation $|S^m||\mathrm{GL}(n,k)|$. The key spaces K' and K'' correspond to key space reductions via sustaining transformations using the estimates from [WP05] and Theorem 2.2.4 respectively.

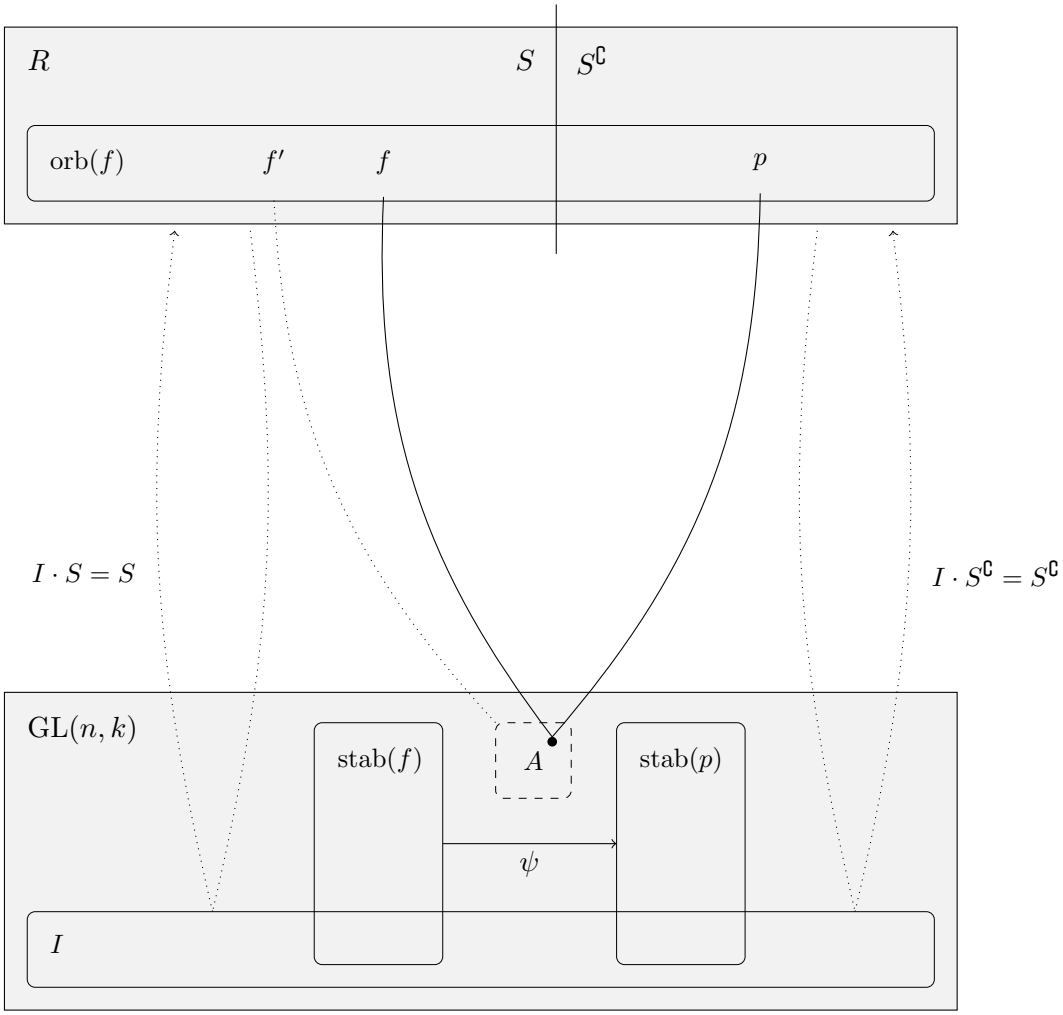

Figure 2.1: Key space of UOV. A public key $p \in S^{\mathbb{C}}$ is a composition of $f \in S$ with a secret transformation $A \in \mathrm{GL}(n,k)$. The subset of $\mathrm{GL}(n,k)$ that contains A (dashed) contains other solutions to PEP. The map ψ is the inner automorphism from Lemma 2.3.4. Note that I preserves all central maps by definition, but a particular central map f might have stabilizers outside of I.

m	n	q	$\log_2 \mathsf{P}[g \in S]$	$\log_2 \mathsf{P}[f \circ A \in S]$
2	5	2	-3	-7
2	5	31	-14	-29
44	103	256	$-7\,920$	$-20\,768$
48	144	256	$-9\,408$	$-36\,864$
64	192	2	$-2\,080$	$-8\,193$

Table 2.4: Probability estimates. We denote by $\mathsf{P}[g \in S]$ the probability of a randomly chosen polynomial g in R to be a central map. Column $\mathsf{P}[f \circ A \in S]$ specifies the probability that a randomly chosen matrix $A \in \mathrm{GL}(n, k)$ maps a central map f back to some other central map as specified in Lemma 2.2.5.

Chapter 3

Polynomial Systems of Equations

After a brief introduction to Gröbner bases, we will study complexity estimates, restricting our attention to case $n = m$. While such restriction is artificial, it allows us to produce a self-contained exposition of this topic. We conclude this chapter with two additional algorithms that are of interest to multivariate public key cryptography. We study Thomae-Wolf algorithm in the case $n > 2m$ and the XL method when $n < m$.

3.1 Solving Equations Using Gröbner Bases

Let I be an ideal in the polynomial ring $k[x_1, \ldots, x_n]$. By the Hilbert Basis Theorem [CLO15, Theorem 2.5.4] all ideals in a polynomial ring are finitely generated, so there are $f_1, \ldots, f_m \in k[x_1, \ldots, x_n]$ such that

$$I = \langle f_1, \ldots, f_m \rangle = \left\{ \sum_{i=1}^{m} f_i h_i \mid h_i \in k[x_1, \ldots, x_n] \right\}.$$

Informally, a Gröbner basis of an ideal I is a finite set of generators $G \subseteq I$ such that division of any element in $k[x_1, \ldots, x_n]$ by G produces a unique remainder irrespective of how the generators are ordered during division[1].

Before we formally define a Gröbner basis we need to review the ordering of terms in polynomials. The following definition plays a crucial role in all that follows. Some statements below may even assume that a monomial order is fixed without stating it explicitly.

[1] The equivalence of this definition to the one given below follows from the list of 10 characterizations of Gröbner bases provided by Becker and Weispfenning [BW93, Theorem 5.35].

Definition 3.1.1 (Monomial Order)**.** A monomial order is a relation $<$ on $\mathbb{Z}^n_{\geq 0}$ such that $<$ is a total order, a well-ordering, and satisfies the following implication for all $\gamma \in \mathbb{Z}^n_{\geq 0}$

$$\alpha < \beta \implies \alpha + \gamma < \beta + \gamma. \tag{3.1}$$

While the monomial order is defined on $\mathbb{Z}^n_{\geq 0}$ we will use it on monomials by means of the following map. Recall from Section 2.1 that we denote by $M(x, n)$ the set of all monomials in $k[x_1, \ldots, x_n]$ and abbreviate it as M when the names and number of the variables are clear. Let $\varphi : M \to \mathbb{Z}^n_{\geq 0}$ be defined by

$$\varphi(x_1^{\alpha_1} \cdots x_n^{\alpha_n}) = (\alpha_1, \ldots, \alpha_n).$$

The map φ is a bijection and it preserves multiplication of monomials via $\varphi(x^\alpha x^\beta) = \varphi(x^\alpha) + \varphi(x^\beta) = \alpha + \beta$. Consequently, we will write $x^\alpha < x^\beta$ to mean $\varphi(x^\alpha) < \varphi(x^\beta)$. Observe that φ enables us to identify multiplication in M with addition in $\mathbb{Z}^n_{\geq 0}$, which is particularly convenient for divisibility of monomials

$$x^\alpha \mid x^\beta \iff \beta = \alpha + \gamma \text{ for some } \gamma \in \mathbb{Z}^n_{\geq 0}.$$

Once the monomial order is fixed we can refer to leading terms of polynomials in $k[x_1, \ldots, x_n]$ without ambiguity. Let $f = \sum_{\alpha \in A} c_\alpha x^\alpha$ be a nonzero polynomial in $k[x_1, \ldots, x_n]$ and denote by $\beta = \max\{|\alpha| : \alpha \in A\}$. We define the leading monomial (LM), the leading coefficient (LC), and the leading term (LT) of f by

$$\mathrm{LM}(f) = x^\beta, \qquad \mathrm{LC}(f) = c_\beta, \qquad \mathrm{LT}(f) = \mathrm{LC}(f)\mathrm{LM}(f).$$

The purpose of property (3.1) should now become apparent. It ensures that monomial ordering is compatible with multiplication in $k[x_1, \ldots, x_n]$. That is, $\mathrm{LT}(fg) = \mathrm{LT}(f)\mathrm{LT}(g)$.

The standard monomial orders are lexicographic ($<_{\mathrm{lex}}$), degree lexicographic ($<_{\mathrm{deglex}}$), reverse lexicographic ($<_{\mathrm{revlex}}$), and degree reverse lexicographic ($<_{\mathrm{degrevlex}}$) defined by

$$\begin{aligned}
\alpha <_{\mathrm{lex}} \beta &\overset{\text{def.}}{\iff} \text{the leftmost nonzero component of } (\beta - \alpha) \text{ in } \mathbb{Z}^n \text{ is positive,} \\
\alpha <_{\mathrm{deglex}} \beta &\overset{\text{def.}}{\iff} |\alpha| < |\beta| \text{ or } \left(|\alpha| = |\beta| \text{ and } \alpha <_{\mathrm{lex}} \beta\right), \\
\alpha <_{\mathrm{revlex}} \beta &\overset{\text{def.}}{\iff} \text{the rightmost nonzero component of } (\beta - \alpha) \text{ in } \mathbb{Z}^n \text{ is negative,} \\
\alpha <_{\mathrm{degrevlex}} \beta &\overset{\text{def.}}{\iff} |\alpha| < |\beta| \text{ or } \left(|\alpha| = |\beta| \text{ and } \alpha <_{\mathrm{revlex}} \beta\right),
\end{aligned}$$

Any monomial order that satisfies $\alpha < \beta$ whenever $|\alpha| < |\beta|$ is called a *graded order*.

Definition 3.1.2 (Gröbner Basis)**.** Fix a monomial order and let $I \neq \langle 0 \rangle$ be an ideal in $k[x_1, \ldots, x_n]$. A finite subset G of I is said to be a Gröbner basis for I if

$$\langle \{\mathrm{LT}(f) \mid f \in I\} \rangle = \langle \{\mathrm{LT}(g) \mid g \in G\} \rangle .$$

We will denote $\langle \{\mathrm{LT}(h) \mid h \in H\} \rangle$ by $\langle \mathrm{LT}(H) \rangle$ for all nonempty subsets H of $k[x_1, \ldots, x_n]$, so the Gröbner basis condition can be restated as $\langle \mathrm{LT}(I) \rangle = \langle \mathrm{LT}(G) \rangle$. Note that we have $\langle \mathrm{LM}(H) \rangle = \langle \mathrm{LT}(H) \rangle$ since k is a field.

There is a certain kind of a Gröbner basis that is unique for each ideal and monomial order by [CLO15, Theorem 2.7.5].

Definition 3.1.3 (Reduced Gröbner Basis)**.** A finite subset G of an ideal I in $k[x_1, \ldots, x_n]$ is said to be a reduced Gröbner basis for I if

(i) G is a Gröbner basis for I with respect to some monomial ordering
(ii) For all $g \in G$, $\mathrm{LC}(g) = 1$
(iii) For all $g = \sum_{\alpha \in A} c_\alpha x^\alpha \in G$ and all $\alpha \in A$, $x^\alpha \notin \langle \mathrm{LT}(G \setminus \{g\}) \rangle$.

We need one more definition before we can discuss the use of Gröbner bases for the purposes of solving multivariate systems of equations.

Definition 3.1.4. (Affine Variety and Ideal of a Variety) Let $I \subseteq k[x_1, \ldots, x_n]$ be an ideal. A subset $\mathbb{V}(I) \subseteq k^n$ is said to be an affine variety of I if

$$\mathbb{V}(I) = \{a \in k^n \mid f(a) = 0 \text{ for all } f \in I\}.$$

Let $V \subseteq k^n$ be an affine variety. We define the ideal $\mathbb{I}(V) \subseteq k[x_1, \ldots, x_n]$ to be

$$\mathbb{I}(V) = \{f \in k[x_1, \ldots, x_n] \mid f(a) = 0 \text{ for all } a \in V\}.$$

Suppose we are given a consistent system of equations $f_1, \ldots, f_m$ in $k[x_1, \ldots, x_n]$ and we want to find a point $a \in \mathbb{V}(f_1, \ldots, f_m)$. We start with computing a Gröbner basis for $I = \langle f_1, \ldots, f_m \rangle$ in lexicographic order, which is particularly convenient for solving polynomial systems of equations.

Theorem 3.1.5 (Elimination Theorem)**.** *Let G be a Gröbner basis of an ideal I in $k[x_1, \ldots, x_n]$ with respect to lexicographic ordering. Then*

$$G \cap k[x_i, \ldots, x_n] = I \cap k[x_i, \ldots, x_n],$$

for all $i = 1, \ldots, n$.

Proof. [CLO15, Theorem 3.1.2] □

The set $I \cap k[x_{i+1}, \ldots, x_n]$ forms an ideal in $k[x_{i+1}, \ldots, x_n]$ that is called the *i-th elimination ideal*. If $G \cap k[x_n] \neq \emptyset$ then we expect to find univariate polynomials in the intersection. These polynomials can be factored using Berlekamp's algorithm [Ber67] to find a set of solutions $a_n \in k$. Once a solution a_n is obtained, we would like to extend it to (a_{n-1}, a_n). The existence of such extensions is established by the following theorem.

Theorem 3.1.6 (Extension Theorem)**.** *Suppose k is algebraically closed. Let $I = \langle f_1, \ldots, f_m \rangle$ be an ideal in $k[x_1, \ldots, x_n]$ and let $J = I \cap k[x_2, \ldots, x_n]$ be the first elimination ideal of I. Denote by $d_i \in \mathbb{Z}^n_{\geq 0}$ the maximum degree of x_1 among all monomials in f_i and write the generators of I as*

$$f_i = \sum_{j=0}^{d_i} h_{i,j} x_1^j$$

for some $h_{i,j} \in k[x_2, \ldots, x_n]$. Then $a' = (a_2, \ldots, a_n) \in \mathbb{V}(J)$ and $a' \notin \mathbb{V}(h_{1,d_1}, \ldots, h_{m,d_m})$ imply the existence of $a_1 \in k$ such that $(a_1, a') \in \mathbb{V}(I)$.

Proof. [CLO15, Theorem 3.6.8]. □

These theorems produce an algorithm for solving polynomial systems of equations. Let $I = \langle f_1, \ldots, f_m \rangle$ be an ideal in $k[x_1, \ldots, x_n]$ and suppose k is algebraically closed. We compute a Gröbner basis G for I in lexicographic order. Starting with an empty solution, as i ranges from n to 1 we maintain an intermediate solution from the previous step $(a_{i+1}, \ldots, a_n) \in \mathbb{V}(I \cap k[x_{i+1}, \ldots, x_n])$. At each step i we proceed as follows:

1. Denote by $\{g_1, \ldots, g_t\}$ the intersection $G \cap k[x_i, \ldots, x_n]$ and substitute the intermediate solution to obtain $g'_j = g_j(a_{i+1}, \ldots, a_n) \in k[x_i]$.
2. Solve the univariate system by factoring the greatest common divisor of $g'_1, \ldots, g'_t$.
3. Check the solutions $a_i \in k$ using the extension theorem:
   (a) If there is a solution, we update the intermediate solution to $(a_i, a_{i+1}, \ldots, a_n)$ and continue to $i + 1$.
   (b) If there are no solutions, we choose an alternative intermediate solution $(a_{i+1}, \ldots, a_n)$ or terminate if there are none.

For finite fields, we can work over an algebraic closure of k and exclude the solutions of no interest by adding field equations into a system at Step 2.

3.2 Complexity of Gröbner Basis Computation

The goal of time complexity estimation is to express the number of operations required to compute a Gröbner basis as a function of input parameters, such as the number of variables n, the number of polynomials m, and the order of the field k. The operations are counted in

the base field k and reported using the big O notation. We will develop the estimates from the ground up.

3.2.1 Hilbert Polynomial of a Monomial Ideal

Gröbner basis complexity estimation uses Hilbert functions, Hilbert polynomials, and other objects defined in the following sections. In this section we will prove the existence and uniqueness of Hilbert polynomials for the case of monomial ideals (Theorem 3.2.4). The definition of a Hilbert polynomial for an arbitrary ideal is postponed until the next section.

Recall that a *monomial ideal* is an ideal generated by monomials. If $I = \langle \{x^\alpha \mid \alpha \in A \subseteq \mathbb{Z}^n_{\geq 0}\} \rangle$ is such an ideal, we can test whether a monomial is in I using divisibility

$$x^\beta \in I \iff x^\alpha \mid x^\beta \text{ for some } \alpha \in A.$$

Definition 3.2.1 (Complement of Monomial Ideal)**.** Let I be a monomial ideal in $k[x_1, \ldots, x_n]$.

(i) A subset $C(I) \subseteq \mathbb{Z}^n_{\geq 0}$ is called a complement of the monomial ideal I if

$$C(I) = \{\alpha \in \mathbb{Z}^n_{\geq 0} \mid x^\alpha \notin I\}.$$

(ii) Let $E \subseteq \{1, \ldots, n\}$. We say that $[E]$ is a coordinate subspace of $\mathbb{Z}^n_{\geq 0}$ if

$$[E] = \{(a_1, \ldots, a_n) \in \mathbb{Z}^n_{\geq 0} \mid a_i = 0 \text{ for all } i \notin E\}.$$

(iii) A subset $T \subseteq \mathbb{Z}^n_{\geq 0}$ is called a translate of the coordinate subspace $[E]$ if

$$T = \alpha + [E] = \{\alpha + \beta \in \mathbb{Z}^n_{\geq 0} \mid \beta \in [E]\},$$

where $\alpha \in \mathbb{Z}^n_{\geq 0}$ is such that the dot product $\alpha \cdot \beta = 0$ for all $\beta \in [E]$.

The following lemma summarizes the main properties of the complements.

Lemma 3.2.2. *Let $I \subseteq k[x_1, \ldots, x_n]$ be a proper monomial ideal, $E \subseteq \{1, \ldots, n\}$, and $\alpha = \beta + \gamma$ in $\mathbb{Z}^n_{\geq 0}$. Then*

(i) $\mathbb{V}(\{x_i \mid i \in E\}) \subseteq \mathbb{V}(I) \iff [E^{\complement}] \subseteq C(I)$,
(ii) $\alpha + [E] \subseteq C(I) \implies \beta + [E] \subseteq C(I)$.

Proof. (i) [CLO15, Proposition 9.2.2.(i)]

(ii) Suppose $\beta + [E] \not\subseteq C(I)$, then there must be $\delta \in [E]$ such that $x^{\beta+\delta} \in I$. Since I is an ideal, we can multiply by x^γ to obtain $x^\gamma x^{\beta+\delta} = x^{\alpha+\delta} \in I$, which contradicts that $\alpha + [E] \subseteq C(I)$. □

A variety of a monomial ideal can be written as a finite union of linear subspaces in k^n. Suppose $I \subseteq k[x_1, \ldots, x_n]$ is a proper monomial ideal. We can write $I = \langle x^{\alpha(1)}, \ldots, x^{\alpha(r)} \rangle$ for some $\alpha(i) \in \mathbb{Z}^n_{\geq 0}$. Using the fact that $\mathbb{V}(I + J) = \mathbb{V}(I) \cap \mathbb{V}(J)$, we rewrite it as a finite intersection

$$\mathbb{V}(I) = \mathbb{V}(x^{\alpha(1)}) \cap \cdots \cap \mathbb{V}(x^{\alpha(r)}).$$

Since $\mathbb{V}(IJ) = \mathbb{V}(I) \cup \mathbb{V}(J)$, each $\mathbb{V}(x^{\alpha(i)})$ can further be broken down into a finite union

$$\mathbb{V}(x_{i_1}^{\alpha_{i_1}} \cdots x_{i_s}^{\alpha_{i_s}}) = \mathbb{V}(x_{i_1}^{\alpha_{i_1}}) \cup \cdots \cup \mathbb{V}(x_{i_s}^{\alpha_{i_s}}),$$

where $\mathbb{V}(x_{i_j}^{\alpha_{i_j}}) = \mathbb{V}(x_{i_j})$ by $\mathbb{V}(I) = \mathbb{V}(\sqrt{I})$. Applying the distributivity property of unions with respect to intersections, we rewrite the original variety as

$$\mathbb{V}(I) = \mathbb{V}(\{x_i \mid i \in A_1\}) \cup \cdots \cup \mathbb{V}(\{x_i \mid i \in A_r\}),$$

for some $A_j \subseteq \{1, \ldots, n\}$. Next, we eliminate all $\mathbb{V}(\{x_i \mid i \in A_p\}) \subseteq \mathbb{V}(\{x_i \mid i \in A_q\})$ for $p \neq q$ to obtain the unique minimal decomposition of $\mathbb{V}(I)$ into irreducible varieties. Observe that varieties $\mathbb{V}(\{x_i \mid i \in A_j\})$ form linear subspaces in k^n. This motivates the following definition

$$\dim \mathbb{V}(\{x_i \mid i \in A_j\}) = n - |A_j|.$$

Similarly, we can break down the complement of the monomial ideal $C(I)$ into translates of the coordinate subspaces of $\mathbb{Z}^n_{\geq 0}$.

Theorem 3.2.3. *If $I \subseteq k[x_1, \ldots, x_n]$ is a proper monomial ideal, then there are finitely many translates $T_1, \ldots, T_r \in \mathbb{Z}^n_{\geq 0}$ such that $C(I) = T_1 \cup \cdots \cup T_r$*

Proof. [CLO15, Theorem 9.2.3]. □

We are now ready to state the main result.

Theorem 3.2.4. *Let $I \subseteq k[x_1, \ldots, x_n]$ be a proper monomial ideal and let $d \in \mathbb{Z}_{\geq 0}$ be the maximal dimension of the linear subspace in $\mathbb{V}(I)$. Denote by $C_{\leq t}(I)$ the exponents of monomials not in I of degree at most t.*

(i) There is $p \in \mathbb{Q}[t]$ of degree d such that $p(t) = |C_{\leq t}(I)|$ for all $t \geq t_0$ in $\mathbb{Z}_{\geq 0}$.
(ii) The polynomial p can be written as

$$p = \sum_{i=0}^{d} a_i \binom{t}{i},$$

where the coefficients $a_i \in \mathbb{Z}$ are unique and $a_d > 0$.

Proof. By Theorem 3.2.3 we can write $C(I) = T^{(1)} \cup \cdots \cup T^{(r)}$ for some translates $T^{(i)} \subseteq \mathbb{Z}^n_{\geq 0}$ such that $T^{(i)} \not\subseteq T^{(j)}$ for $i \neq j$. If we restrict to monomials of degree at most t, we also have

$$C_{\leq t}(I) = T^{(1)}_{\leq t} \cup \cdots \cup T^{(r)}_{\leq t},$$

where $T^{(i)}_{\leq t}$ are subsets of the respective translates with the monomial degrees restricted accordingly. To count the elements in $C_{\leq t}(I)$ we apply the Inclusion-Exclusion Principle

$$|C_{\leq t}(I)| = \sum_{i=1}^{r} \left| T^{(i)}_{\leq t} \right| + \sum_{i=2}^{r} \left((-1)^{i-1} \sum_{1 \leq j_1 < \cdots < j_i \leq r} \left| T^{(j_1)}_{\leq t} \cap \cdots \cap T^{(j_i)}_{\leq t} \right| \right). \tag{3.2}$$

However, first we need to develop several auxiliary results. We define the dimension of $T^{(j)}$, denoted $\dim T^{(j)}$, to mean the number of components in the span of the respective coordinate subspace in $\mathbb{Z}^n_{\geq 0}$.

Claim 1: Each $T^{(j)}$ is of dimension at most d with at least one translate of dimension equal to d. Suppose there is $T^{(\ell)} = \alpha + [A_\ell]$ such that $\dim T^{(\ell)} > d$. By Lemma 3.2.2.(ii), $[A_\ell] \subseteq C(I)$ and by Lemma 3.2.2.(i), there is some linear subspace $V = \mathbb{V}(\{x_i \mid i \in A^{\complement}_\ell\})$ in $\mathbb{V}(I)$. Then $\dim V = n - |A^{\complement}_\ell| = |A_\ell| > d$, which contradicts that d is the maximal dimension of the linear subspace in $\mathbb{V}(I)$. Let $P = \mathbb{V}(\{x_i \mid i \in E\})$ be the linear subspace in $\mathbb{V}(I)$ of maximal dimension, then $\dim P = n - |E| = d$. Applying Lemma 3.2.2.(i) implies that there is $[E^{\complement}]$ of dimension $|E^{\complement}| = n - |E| = d$ in $C(I)$.

Claim 2: If $A = \alpha + [U]$ and $B = \beta + [V]$ are such that $A \neq B$, then $\dim A \cap B < \max\{\dim A, \dim B\}$. If $A \cap B = \emptyset$, the dimension of $A \cap B$ is zero, which is strictly less than the maximum of $\dim A$ and $\dim B$ because $A \neq B$. Suppose that $A \cap B \neq \emptyset$. If $[U] = [V]$, then $A \neq B$ implies $\alpha \neq \beta$, but then $A \cap B = \emptyset$ because α and β are orthogonal to $[U] = [V]$. It must be that $[U] \neq [V]$ and hence $\dim [U] \cap [V] < \max\{\dim [U], \dim [V]\}$. Fix a monomial order and let $\gamma = \min\{A \cap B\}$, then $A \cap B = \gamma + [U] \cap [V]$. Therefore, $\dim A \cap B = \dim [U] \cap [V] < \max\{\dim [U], \dim [V]\} = \max\{\dim A, \dim B\}$.

Claim 3: If $T = \alpha + [E]$, then there is $g \in \mathbb{Q}[t]$ of degree $e = \dim [E]$ with $\mathrm{LC}(g) > 0$ such that $g(t) = |T_{\leq t}|$ for all $t \geq |\alpha|$. Observe that $e = |E|$. The number of points $\beta \in [E]$ satisfying $|\beta| \leq t$ is $\binom{e+t}{t}$ by Lemma 2.1.1. Accounting for translation by α and expanding the binomial coefficient, the number of points in T corresponding to monomials of degree at most t is given by

$$g(t) = \binom{e + t - |\alpha|}{t - |\alpha|} = \frac{1}{e!} t^e + q(t),$$

whenever $t \geq |\alpha|$. Note that $q \in \mathbb{Q}[t]$ is such that $\deg(q) < e$ and therefore $\mathrm{LC}(g) > 0$.

(i) Consider the equation (3.2). By Claim 1, the first sum on the right-hand side

$$g(t) = \sum_{i=1}^{r} \left| T_{\leq t}^{(i)} \right|$$

contains at least one translate of dimension d and all other translates are of dimension $\leq d$. By Claim 3, each $|T_{\leq t}^{(i)}|$ is a polynomial in $\mathbb{Q}[t]$ of degree $\leq d$ and the leading coefficients are positive. Consequently, no cancellations of leading terms may occur in g. We thus obtain $g \in \mathbb{Q}[t]$, $\deg(g) = d$, and $\mathrm{LC}(g) > 0$.

Bringing our attention to the second sum on the right-hand side of equation (3.2)

$$h(t) = \sum_{i=2}^{r} \left((-1)^{i-1} \sum_{1 \leq j_1 < \cdots < j_i \leq r} \left| T_{\leq t}^{(j_1)} \cap \cdots \cap T_{\leq t}^{(j_i)} \right| \right),$$

we see that each $T = T^{(j_1)} \cap \cdots \cap T^{(j_i)}$ is an intersection of at least two distinct translates. Therefore, by Claim 2, $\dim T < \max\{\dim T^{(j_1)}, \ldots, \dim T^{(j_i)}\} \leq d$. Invoking Claim 3 again, we see that the polynomials counting $|T_{\leq t}|$ for t large enough, must all be of degree strictly less than d. Therefore

$$p(t) = g(t) + h(t) \in \mathbb{Q}[t]$$

is such that $\deg(p) = d$, $\mathrm{LC}(p) = \mathrm{LC}(g) > 0$, and $p(t) \in \mathbb{Z}$ for all $t \geq t_0$ in $\mathbb{Z}_{\geq 0}$.

(ii) Let $b_0 = p(t_0), b_1 = p(t_0+1), \ldots, b_d = p(t_0+d) \in \mathbb{Z}_{\geq 0}$ and construct a Newton-Gregory interpolating polynomial $q \in \mathbb{Q}[t]$ using

$$q = \sum_{i=0}^{d} \Delta_i \binom{t}{i}, \qquad \Delta_i = \sum_{j=0}^{i} (-1)^j \binom{i}{j} b_{i-j},$$

so that $q(i) = b_i = p(t_0 + i)$ for all $i = 0, \ldots, d$. Since $\binom{i}{j} \in \mathbb{Z}$ and $b_{i-j} \in \mathbb{Z}$, each Δ_i is also in $\mathbb{Z}$. Polynomials $\binom{t}{i} \in \mathbb{Q}[t]$ are of degree i and $\binom{t}{i}(\mathbb{Z}) \subseteq \mathbb{Z}$. Therefore, q satisfies $q(\mathbb{Z}) \subseteq \mathbb{Z}$ and has degree at most d. Translating by t_0, we obtain a polynomial $p' \in \mathbb{Q}[t]$ defined by

$$p' = \sum_{i=0}^{d} \Delta_i \binom{t - t_0}{i}.$$

We claim that $p = p'$ in $\mathbb{Q}[t]$. Suppose $p - p' \neq 0$ in $\mathbb{Q}[t]$. Since $(p - p')(t_0 + i) = 0$ for $i = 0, \ldots, d$, the polynomial $p - p'$ must have at least $d + 1$ roots. This implies that $\deg(p - p') \geq d + 1$, which contradicts that $p - p'$ has degree at most d.

Since p is merely a translation of q by an integer t_0, $q(\mathbb{Z}) \subseteq \mathbb{Z}$ implies $p(\mathbb{Z}) \subseteq \mathbb{Z}$. We repeat the process of constructing a Newton-Gregory interpolating polynomial, but this time starting with $c_0 = p(0), \ldots, c_d = p(d)$. By a similar argument, we obtain

$$p = \sum_{i=0}^{d} a_i \binom{t}{i}, \tag{3.3}$$

where a_i are in $\mathbb{Z}$ because they correspond to Δ_i in the argument above and $a_d > 0$ by part (i). The expression (3.3) is unique because polynomials $\binom{t}{i}$ are of degree i and so they are linearly independent. □

3.2.2 Index of Regularity

Index of regularity plays an important role in complexity estimation. However, it depends on a number of auxiliary definitions, which we provide below. While our main interest is in homogeneous systems, in this section we will temporarily work with both affine and homogeneous cases. Theorem 3.2.9 enables transition between these cases.

The polynomial ring $R = k[x_1, \ldots, x_n]$ forms a vector space over k spanned by monomials $M(x, n)$. If we restrict the degree of polynomials, we obtain a finite dimensional vector space

$$R_{\leq t} = \operatorname{span} M_{\leq t}(x, n) = \{f \in k[x_1, \ldots, x_n] \mid \deg(f) \leq t\} \cup \{0\}.$$

The dimension of $R_{\leq t}$, as a vector space over k, is given by

$$\dim_k R_{\leq t} = |M_{\leq t}(x, n)|.$$

An ideal I in R forms a vector subspace of R because ideals are closed under addition and multiplication by elements of $R \supseteq k$. This enables us to define

$$I_{\leq t} = I \cap R_{\leq t},$$

which is a vector subspace of $R_{\leq t}$, because multiplication by scalars and addition of polynomials do not increase the degree.

For the homogeneous polynomials and ideals, we denote by R^h the polynomial ring $k[x_0, \ldots, x_n]$ of $n + 1$ variables over k. The corresponding degree restrictions are defined by

$$\begin{aligned} R^h_t &= \{f \in R^h \mid f \text{ is homogeneous of degree } t\} \cup \{0\}, \\ I_t &= I \cap R^h_t, \end{aligned}$$

where I is assumed to be a homogeneous ideal. Because addition of homogeneous polynomials of degree t produces a homogeneous polynomial of degree t or zero, the sets R^h_t and I_t form the respective vector subspaces. Consequently, we can also talk about quotient subspaces and their dimension.

Definition 3.2.5 (Hilbert Function)**.** Let I be an ideal in R. We say that $f : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ is an affine Hilbert function of R/I if

$$f(t) = \dim_k \left(R_{\leq t} / I_{\leq t} \right).$$

If I is a homogeneous ideal in R^h, we define a Hilbert function of R^h/I by $f(t) = \dim_k \left(R^h_t / I_t \right)$.

Since a Hilbert function is always used in the context of a particular polynomial ring and an ideal, we introduce the following notation

$$\begin{aligned} \mathrm{HF}_{R/I}(t) &= \dim_k \left(R_{\leq t} / I_{\leq t} \right), \\ \mathrm{HF}_{R^h/I}(t) &= \dim_k \left(R^h_t / I_t \right), \end{aligned}$$

where the second line (non-affine case) assumes that the ideal I is homogeneous.

To compute a value of the Hilbert function we need to subtract the respective dimensions

$$\mathrm{HF}_{R/I}(t) = \dim_k R_{\leq t} - \dim I_{\leq t}.$$

It is easy to compute $\dim_k R_{\leq t} = |M_{\leq t}(x, n)| = \binom{n-t}{t}$ using Lemma 2.1.1. However, it is not immediately obvious how to handle $\dim_k I_{\leq t}$.

Lemma 3.2.6. *If I is an ideal in R and $\langle \mathrm{LT}(I) \rangle$ is obtained with respect to a graded monomial order, then*

$$\mathrm{HF}_{R/I} = \mathrm{HF}_{R/\langle \mathrm{LT}(I) \rangle}.$$

The same result holds with respect to any monomial order if I is a homogeneous ideal in R^h.

Proof. Affine case [CLO15, Proposition 9.3.4]; homogeneous case [CLO15, Proposition 9.3.9]. □

When $I \subsetneq R$ is a monomial ideal, the corresponding Hilbert function counts the number of elements in the complement

$$\mathrm{HF}_{R/I}(t) = |C_{\leq t}(I)|.$$

Now if I is an ideal in R, which is not necessarily a monomial ideal, we can fix a graded monomial order such as $<_{\mathrm{deglex}}$ and compute a monomial ideal $\langle \mathrm{LT}(I) \rangle$. Then, By Lemma 3.2.6, we have

$$\mathrm{HF}_{R/I}(t) = |C_{\leq t}(\langle \mathrm{LT}(I) \rangle)|.$$

If we now invoke Theorem 3.2.4, we are assured that there is a unique polynomial $p \in \mathbb{Q}[t]$ and $t_0 \in \mathbb{Z}_{\geq 0}$ such that $p(t)$ counts the number of monomials not in $\langle \mathrm{LT}(I) \rangle$ of degree at most t for all $t \geq t_0$. Note that the Hilbert polynomial of R/I is uniquely determined for each monomial ideal and a monomial ideal, in turn, has a unique minimal basis [CLO15, Proposition 2.4.7].

Definition 3.2.7 (Hilbert Polynomial)**.** Let I be an ideal in R. We say that $p \in \mathbb{Q}[t]$ is an affine Hilbert polynomial of R/I if there is $t_0 \in \mathbb{Z}_{\geq 0}$ such that

$$p(t) = \mathrm{HF}_{R/I}(t),$$

for all $t \geq t_0$. If I is a homogeneous ideal in R^h, we require that $p(t) = \mathrm{HF}_{R^h/I}(t)$ for all $t \geq t_0$.

Similarly to Hilbert functions, we denote an affine Hilbert polynomial and a Hilbert polynomial by $\mathrm{HP}_{R/I}$ and $\mathrm{HP}_{R^h/I}$ respectively with the assumption that the ideal I in $\mathrm{HP}_{R^h/I}$ is homogeneous. We can now define the primary object of this section.

Definition 3.2.8 (Index of Regularity)**.** Let I be an ideal in R. An integer $t_0 \in \mathbb{Z}_{\geq 0}$ is called the index of regularity of I if t_0 is the smallest integer such that

$$\mathrm{HF}_{R/I}(t) = \mathrm{HP}_{R/I}(t),$$

for all $t \geq t_0$.

The following results allows us to pass between homogeneous and inhomogeneous cases when working with Hilbert functions and polynomials.

Theorem 3.2.9. *Let I be an ideal in R and suppose I^h is its homogenization in R^h. Then*

$$\mathrm{HF}_{R/I}(t) = \mathrm{HF}_{R^h/I^h}(t) \qquad \textit{and} \qquad \mathrm{HP}_{R/I}(t) = \mathrm{HP}_{R^h/I^h}(t),$$

for all $t \in \mathbb{Z}_{\geq 0}$.

Proof. [CLO15, Theorem 9.3.12]. □

We define the dimension of a variety as follows.

Definition 3.2.10 (Dimension of a Variety)**.** The dimension of a nonempty affine variety $V \subseteq k^n$ is the degree of the Hilbert polynomial of the ideal $\mathbb{I}(V)$. Similarly, the dimension of a nonempty projective variety $W \subseteq \mathbb{P}^n(k)$ is defined by $\dim\, W = \deg\left(\mathrm{HP}_{R^h/\mathbb{I}(W)}\right)$.

3.2.3 Gröbner Basis up to Degree d

Instead of developing a complete Gröbner basis algorithm with the corresponding termination conditions, we can obtain results about complexity of computing Gröbner bases using simpler methods developed in this section.

To streamline the discussion below we will introduce the following two definitions. We will denote by $\mathrm{MC}(f, x^\alpha)$ the monomial coefficient of x^α in a polynomial f.

Definition 3.2.11 (Macaulay Matrix)**.** Fix a monomial order and let $f_1, \ldots, f_m \in k[x_1, \ldots, x_n]$. Set $d = \max\{\deg(f_i) \mid 1 \leq i \leq m\}$. We say that $\mathbb{M}(f_1, \ldots, f_m) = (c_{i,j})$ is a Macaulay matrix of $f_1, \ldots, f_m$ if

$$c_{i,j} = \mathrm{MC}(f_i, x^{\alpha(j)}),$$

as $\alpha(j)$ ranges over $M_{\leq d}(x, n) \setminus \{1\}$ in decreasing order of the specified monomial order. The dimensions of $\mathbb{M}(f_1, \ldots, f_m)$ are $m \times (|M_{\leq d}(x, n)| - 1)$.

If the polynomials $f_1, \ldots, f_m$ are homogeneous, we assume that the matrix $\mathbb{M}(f_1, \ldots, f_m)$ is in $k^{m \times |M_d|}$ and $\alpha(j)$ ranges over M_d as opposed to $M_{\leq d}$. We now extend the definition of the Macaulay matrix to take into account all products $x^\alpha f_i$ such that $\deg(x^\alpha f_i) = d$ for some fixed degree d.

Definition 3.2.12 (Macaulay Matrix of Degree d)**.** Fix a monomial order and let $f_1, \ldots, f_m \in k[x_1, \ldots, x_n]$ be homogeneous polynomials. We say that $\mathbb{M}_d(f_1, \ldots, f_m)$ is a Macaulay matrix of degree $d \in \mathbb{N}$ if it is a Macaulay matrix $\mathbb{M}(f)$ of an ordered sequence of homogeneous polynomials f defined by

$$f = (x^\alpha f_i \mid (\alpha, i) \in F_d),$$

where the index set F_d is defined by

$$F_d = \{(\alpha, i) \mid 1 \leq i \leq m,\ e = d - \deg(f_i) > 0,\ \alpha \in M_e\}.$$

The order in which the index set F_d is enumerated defines the order of rows in the Macaulay matrix. We assume that this order is fixed to some enumeration of F_d so that we can read the polynomials corresponding to rows of $\mathbb{M}_d(f_1, \ldots, f_r)$ by specifying the row index (α, i) in F_d.

On the other hand, the order of columns in a Macaulay matrix is defined by listing monomials in M_d in decreasing order with respect to the fixed monomial order. This plays an important role in the results below.

Theorem 3.2.13. *Let $I = \langle f_1, \ldots, f_m \rangle$ be an ideal in $k[x_1, \ldots, x_n]$ generated by homogeneous polynomials f_i. Then the nonzero rows of the echelon form of $\mathbb{M}_d(f_1, \ldots, f_m)$ yield a vector space basis of I_d for all $d \in \mathbb{Z}_{\geq 0}$.*

Proof. Fix $d \in \mathbb{Z}_{\geq 0}$ and re-index the polynomials $f_1, \ldots, f_m$ so that $\deg(f_1) < \deg(f_j)$ whenever $i < j$.

We first claim that the polynomials $x^\alpha f_i$ corresponding to the rows of $\mathbb{M}_d(f_1, \ldots, f_m)$ span I_d. Let $f \in I_d$, then f is homogeneous of degree d and

$$f = f_1 g_1 + \cdots f_r g_r$$

for some $g_i \in k[x_1, \ldots, x_n]$ and $r \leq m$ with $\deg(f_r) \leq d$. The polynomials g_i are not necessarily homogeneous. For $i = 1, \ldots, r$, write g_i in terms of its homogeneous components and group the terms of degree different from $e_i = d - \deg(f_i)$ together. That is,

$$g_i = g_{i,e_i} + h_i,$$

where $\deg(g_{i,e_i}) = e_i$ and no term of h_i has degree e_i. Then

$$f = \sum_{i=1}^{r} f_i(g_{i,e_i} + h_i) = \sum_{i=1}^{r} f_i g_{i,e_i} + \sum_{i=1}^{r} f_i h_i.$$

Now each term $f_i h_i$ has degree different from d, for if $\deg(f_i h_i) = d$, then $\deg(h_i) = d - \deg(f_i) = e_i$. Since f if of degree d, we must have $\sum_{i=1}^{r} f_i h_i = 0$. On the other hand, the terms $f_i g_{i,e_i}$ are all of degree d since $e_i + \deg(f_i) = d$. Therefore,

$$f = \sum_{i=1}^{s} f_i g_{i,e_i} = \sum_{(\alpha,i) \in F_d} c_{\alpha,i} x^\alpha f_i$$

for some $c_{\alpha,i} \in k$.

Let $p_1, \ldots p_s \in k[x_1, \ldots x_n]$ be polynomials corresponding to nonzero rows $1, \ldots, s$ of the echelon form of $\mathbb{M}_d(f_1, \ldots f_m)$. We need to show that p_i are linearly independent and that they span I_d.

Since the columns of $\mathbb{M}_d(f_1, \ldots, f_m)$ are ordered in decreasing order and the matrix is in reduced echelon form, we must have $\mathrm{LM}(p_i) > \mathrm{LM}(p_j)$ whenever $i > j$. If $a_1 p_1 + \cdots + a_s p_s = 0$ for some $a_i \in k$, not all zero, then there is the least index j such that $a_j = 0$. Then $\mathrm{LT}(a_j p_j) \neq 0$ and there is no other term to cancel it out. Therefore, $p_1, \ldots, p_s$ are linearly independent.

Let A denote the row-reduced echelon form of $\mathbb{M}_d(f_1, \ldots, f_m)$. Then there are elementary row matrices $E_1, \ldots E_q$ such that

$$\mathbb{M}_d(f_1, \ldots, f_m) = E_q^{-1} \cdots E_1^{-1} A.$$

In other words, the polynomials in $\mathbb{M}_d(f_1, \ldots, f_m)$ are linear combinations of polynomials in A with coefficients in k. That is,

$$x^\alpha f_i = \sum_{j=1}^{s} b_{\alpha,i,j} p_j$$

for all $(\alpha, i) \in F_d$ where $b_{\alpha,i,j} \in k$. Recall that we first showed that $x^\alpha f_i$ span I_d, so if $f \in I_d$, then

$$f = \sum_{(\alpha,i)\in F_d} a_{\alpha,i} x^\alpha f_i = \sum_{j=1}^{s} \left(\sum_{(\alpha,i)\in F_d} a_{\alpha,i} b_{\alpha,i,j} \right) p_j,$$

by distributivity and the fact that the sums are finite. Therefore, $p_1, \ldots, p_s$ form a basis of I_d. □

Definition 3.2.14 (Gröbner Basis of Degree $\leq d$)**.** Fix a monomial order and let I be a homogeneous ideal in $k[x_1, \ldots, x_n]$. A finite subset $G_{\leq d}$ of I is called a Gröbner basis of I up to degree d if for all homogeneous $f \in I$ with $\deg(f) \leq d$ there is a $g \in G_{\leq d}$ such that $\mathrm{LT}(g) \mid \mathrm{LT}(f)$.

Lemma 3.2.15. *Let $f_1, \ldots, f_m \in k[x_1, \ldots, x_n]$ be homogeneous polynomials and let $d \in \mathbb{N}$. For $i = 1, \ldots, d$, denote by $g_{i,1}, \ldots, g_{i,s_i}$ the nonzero polynomials of the echelon form of $\mathbb{M}_i(f_1, \ldots, f_m)$. Then*

$$G_{\leq d} = \bigcup_{i=1}^{d} \{g_{i,j} \mid j = 1, \ldots, s_i\}$$

is a Gröbner basis of $I = \langle f_1, \ldots, f_m \rangle$ up to degree d.

Proof. Fix $d \in \mathbb{N}$ and let $f \in I$ be homogeneous of degree $e \leq d$. Then $f \in I_e$, so we can write

$$f = \sum_{j=1}^{s_e} c_j g_{e,j}, \tag{3.4}$$

for some $c_j \in k$ by Theorem 3.2.13. Assuming the index j runs top-down over nonzero rows of the echelon form of $\mathbb{M}_e(f_1, \ldots, f_m)$, we have $\mathrm{LM}(g_{e,1}) > \cdots > \mathrm{LM}(g_{e,s_e})$. Therefore, the least index $\ell \in \{1, \ldots, s_e\}$ such that $c_\ell \neq 0$ in equation (3.4) must satisfy $\mathrm{LM}(f) = \mathrm{LM}(g_{e,\ell})$. In other words, there is $g_{e,\ell} \in G_{\leq d}$ such that $\mathrm{LT}(g_{e,\ell}) \mid \mathrm{LT}(f)$, which is precisely the definition of a Gröbner basis up to degree d. □

Corollary 3.2.16. *Assume the notation of Lemma 3.2.15. Let H be a reduced Gröbner basis for a homogeneous ideal $I \subseteq k[x_1, \ldots, x_n]$. If $d_{\max} = \max\{\deg(h) \mid h \in H\}$, then $G_{\leq d_{\max}}$ is also a Gröbner basis for I.*

Proof. Since $G_{\leq d_{\max}}$ is a subset of I, $\langle \mathrm{LT}(G_{\leq d_{\max}}) \rangle \subseteq \langle \mathrm{LT}(I) \rangle$, so we only need to prove the opposite inclusion.

The set H consists of homogeneous polynomials, because H is a reduced Gröbner basis and I is a homogeneous ideal [CLO15, Theorem 8.3.2.(iii)]. By Lemma 3.2.15, $G_{\leq d_{\max}}$ is a Gröbner basis for I up to degree $d_{\max}$. Therefore, if $h \in H$, then $h \in I$ and h is homogeneous of degree $\leq d_{\max}$, so there is $g_{\deg(h),\ell} \in G_{\leq d_{\max}}$ such that $\mathrm{LT}(g_{\deg(h),\ell}) \mid \mathrm{LT}(h)$. Therefore, $\langle \mathrm{LT}(I) \rangle = \langle \mathrm{LT}(H) \rangle \subseteq \langle \mathrm{LT}(G_{\leq d_{\max}}) \rangle$. □

3.2.4 Estimating Complexity When $n = m$

We restrict our attention to the case $n = m$. This is required in order to ensure the regularity of the system (Definition 3.2.19), which allows us to connect the results above to the recent results in complexity analysis (Theorem 3.2.21, Lemma 3.2.22).

Corollary 3.2.16 provided means to compute a Gröbner basis of a homogeneous ideal I in $k[x_1, \ldots, x_n]$ using Macaulay matrices. If we know the maximum degree $d_{\max}$ of polynomials in the reduced Gröbner basis of I, all we need is to compute $G_{\leq d_{\max}}$ using the results from the previous section. This produces an immediate result discussed in Lemma 3.2.18.

The main operation involved in computing a Gröbner basis using the linear algebra approach is to bring a matrix into a reduced row echelon form. The complexity of this operation is equivalent to computing an inverse of a matrix.

Lemma 3.2.17 (Complexity of Computing a Matrix Inverse)**.** *If a matrix $A \in k^{n \times n}$ is invertible, its inverse can be obtained in $O(n^\omega)$ operations in the field k for some constant $\omega \in [2, 3)$.*

Proof. Strassen showed that matrix inversion can be performed for $\omega = \log_2 7 \approx 2.8$ [Str69, Fact 4]. The best known result as of this writing is $\omega \approx 2.37286$ due to Alman and Williams [AW20]. □

Lemma 3.2.18. *Let I be a homogeneous ideal in $k[x_1, \ldots, x_n]$ and let $d_{\max} \in \mathbb{N}$ be the maximum degree of the reduced Gröbner basis for I. Then $G_{\leq d_{\max}}$ can be obtained in*

$$\approx O\left(d_{\max}\binom{n + d_{\max} - 1}{d_{\max}}^{\omega}\right)$$

field operations.

Proof. Since I is homogeneous there are homogeneous generators $f_1, \ldots, f_m$. By Corollary 3.2.16 we need to compute $i = 1, \ldots, d_{\max}$ echelon forms of Macaulay matrices $\mathbb{M}_i(f_1, \ldots, f_m)$ to obtain a Gröbner basis for I.

A Macaulay matrix of degree $d_{\max}$ has dimensions less than $|M_{d_{\max}}| \times |M_{d_{\max}}|$ and all other Macaulay matrices involved in the computation are of smaller dimensions. Therefore, by

Lemma 3.2.17, we need $O(\binom{n+d_{\max}-1}{d_{\max}}^{\omega})$ operations in k to compute the reduced row echelon form of $\mathbb{M}_{d_{\max}}$. The result follows since there are $d_{\max}$ such computations involved. □

The next question is how to determine $d_{\max}$ without actually computing a reduced Gröbner basis. We cannot get an exact value, so we will work towards obtaining an upper bound. There is a particular family of polynomial systems that yields good estimates of $d_{\max}$ and is assumed to represent a randomly chosen system of equations.

Definition 3.2.19 (Regular System)**.** A sequence $f_1, \ldots, f_m$ of homogeneous polynomials in $k[x_1, \ldots, x_n]$ is called a regular system if f_i is not a zero divisor in

$$k[x_1, \ldots, x_n] / \langle f_1, \ldots, f_{i-1} \rangle$$

for all $i = 1, \ldots m$.

When working with a system of equations $I = \langle f_1, \ldots, f_m \rangle$ in $R = k[x_1, \ldots, x_n]$, it is convenient to consider the values of the Hilbert function all at once. This is done using the *Hilbert series*, which is a formal power series given by

$$\mathrm{HS}_{R/I}(z) = \sum_{t=0}^{\infty} \mathrm{HF}_{R/I}(t) z^t.$$

Regular systems can be characterized by the form of the respective Hilbert series.

Lemma 3.2.20. *Let $f_1, \ldots, f_m$ be homogeneous polynomials in $R = k[x_1, \ldots, x_n]$.*

(i) The system $f_1, \ldots f_m$ is regular if and only if the Hilbert series have the following form

$$\mathrm{HS}_{R/I}(z) = \frac{\prod_{t=1}^{n} \left(1 - z^{\deg(f_i)}\right)}{(1-z)^n}.$$

(ii) Suppose $m = n$. The system $f_1, \ldots, f_m$ is regular if and only if the Hilbert series is a polynomial in $\mathbb{Q}[z]$.

Proof. [BFS13, Proposition 4]. □

Note that if $m = n$ and the degree of the Hilbert series is d, then $\mathrm{HF}_{R/I}(t) = \mathrm{HP}_{R/I}(t) = 0$ for all $t > d$. Then the Hilbert polynomial $\mathrm{HP}_{R/I} = 0$ in $\mathbb{Q}[t]$ and so the index of regularity is given by $i_{\mathrm{reg}} = d + 1$.

Theorem 3.2.21. *Suppose $m = n$ and let $f_1, \ldots, f_n$ be a regular system of homogeneous polynomials in $k[x_1, \ldots, x_n]$. If G is a reduced Gröbner basis for $I = \langle f_1, \ldots, f_n \rangle$ with respect to $<_{\mathrm{degrevlex}}$, then the maximum degree of a polynomial in G is bounded by*

$$d_{\max} \leq \sum_{i=1}^{n} (\deg(f_i) - 1) + 1.$$

Proof. [BFS13, Corollary 5]. □

The sum from Theorem 3.2.21 is knows as the Macaulay bound. It was introduced by Macaulay [Mac02] as part of his work on resultants and adapted to Gröbner bases in $<_{\text{degrevlex}}$ by Lazard [Laz83].

In cryptographic literature (e.g., [DPS20], [Beu21]), it is common to see complexity estimates of Gröbner bases computations stated in terms of degree of regularity for semi-regular systems, denoted d_{reg} [BFS04]. A semi-regular system is a generalization of a regular system to $m \geq n$; d_{reg} is a generalization of i_{reg} to semi-regular systems. When $m = n$ and the system is of dimension zero, both definitions match[2]. Consequently, we can reuse the estimate for some of the most efficient Gröbner basis algorithms known as the F_5 family[3].

Lemma 3.2.22. *Suppose $m = n$, $I = \langle f_1, \ldots, f_n \rangle$ is a regular system in $k[x_1, \ldots, x_n]$, and $\dim \mathbb{V}(I) = 0$. Then a Gröbner basis for I can be obtained with respect to a fixed monomial order using*

$$O\left(\binom{n + i_{\text{reg}}}{n}^{\omega}\right)$$

operations in k.

Proof. [BFS04, Theorem 7]. □

It is interesting to observe that the complexity estimate presented in Lemma 3.2.18 is relatively close to the result above and that the use of linear algebra in F_5 is indicated by ω.

Example 3.2.23 (Complexity of Solving UOV)**.**
This example was produced by `stats.sage` from Appendix A.4. Consider a public key p_1, p_2 in $k[x_1, \ldots, x_5]$ for $q = 31$. The respective matrices of polynomials p_1 and p_2 are given by

$$P_1 = \begin{bmatrix} 30 & 10 & 12 & 15 & 12 \\ 10 & 14 & 29 & 0 & 6 \\ 12 & 29 & 19 & 17 & 29 \\ 15 & 0 & 17 & 21 & 28 \\ 12 & 6 & 29 & 28 & 17 \end{bmatrix} \qquad P_2 = \begin{bmatrix} 8 & 4 & 0 & 14 & 7 \\ 4 & 7 & 20 & 13 & 24 \\ 0 & 20 & 8 & 16 & 16 \\ 14 & 13 & 16 & 13 & 6 \\ 7 & 24 & 16 & 6 & 29 \end{bmatrix}.$$

Let $I = \langle p_1, p_2 \rangle$. Since $m \neq n$, the estimates from the discussion above cannot be used, so we want to add some constraints on I. The dimension $\dim \mathbb{V}(I) = 3$, so we randomly generate 3

[2] See the note after [BFS04, Definition 5] and observe that the first coefficient ≤ 0 in $\text{HS}_{R/I}$ is the index of regularity. This, in turn, matches the classification of [BFS04, Proposition 6].

[3] There are several variants of F_5. For example, there is the original F_5 introduced by Faugère [Fau02] and Matrix-F_5 of Bardet-Faugère-Salvy [BFS03].

		$\dim \mathbb{V}(I)$			$d_{\max}$		
m	n	min	avg	max	min	avg	max
2	4	2	2.00	2	2	2.97	3
2	6	4	4.00	4	2	2.97	3
3	6	3	3.00	3	3	3.99	4
3	9	6	6.00	6	3	3.97	4
5	10	5	5.00	5	5	5.97	6
5	12	7	7.00	7	5	5.98	6
5	14	9	9.00	9	5	5.95	6
5	15	10	10.00	10	5	5.95	6

Table 3.1: Statistics of UOV public keys. The test runs 100 times for each parameter set m, n with $q = 31$. On each run the test generates a random key pair and computes $\dim \mathbb{V}(I)$ and $d_{\max}$. The ideal I is generated by the public polynomials. The maximum degree of a polynomial in a reduced Gröbner basis $d_{\max}$ is computed with respect to $<_{\text{degrevlex}}$.

linear constraints

$$\begin{aligned} p'_3 &= 27x_1 + 18x_2 + x_3 + 17x_4 + 28x_5 \\ p'_4 &= 5x_1 + 16x_2 + 22x_3 + 19x_4 + 13x_5 \\ p'_5 &= 26x_1 + 12x_2 + 7x_3 + 20x_4 + 13x_5 \end{aligned}$$

and define a new ideal $J = \langle p_1, p_2, p'_3, p'_4, p'_5 \rangle$. The ideal J is homogeneous, $m = n$, and $\dim \mathbb{V}(J) = 0$. By Lemma 3.2.20, we can determine if J is a regular system by considering the form of the Hilbert series

$$\mathrm{HS}_{J/R}(z) = z^2 + 2z + 1.$$

Since the series is a polynomial we conclude that J is a regular system. Furthermore, $\mathrm{HP}_{J/R} = 0$ and so $i_{\text{reg}} = 3$. By Theorem 3.2.21, the maximum degree of the Gröbner basis of J with respect to $<_{\text{degrevlex}}$ should be $d_{\max} = 3$. We confirm this by computing the Gröbner basis explicitly, but omit it below due to long output. The cumulative stats for different parameter sets over 100 test runs are provided in Tables 3.1 and 3.2. The choice of parameters in these tables is motivated by making the experiments easy to reproduce on a modern computer.

3.3 Thomae-Wolf Algorithm $(n > 2m)$

Suppose we are given a system of equations $p = (p_1, \ldots, p_m) \in k[x_1, \ldots, x_n]^m$, where p_i are homogeneous of degree two. Our goal is to find an element $a \in k^n$ such that $p(a) = 0$. Note

m	n	$D_{\max}$	$d_{\max} \neq D_{\max}$	$i_{\text{reg}} \neq D_{\max}$	$\dim \mathbb{V}(J) \neq 0$	$\text{HS}_{R/J} \in \mathbb{Q}[z]$
2	4	3	5	4	4	96
2	6	3	8	5	5	95
3	6	4	12	9	9	91
3	9	4	6	2	2	98
5	10	6	9	6	6	94
5	12	6	5	5	5	95
5	14	6	9	5	5	95
5	15	6	6	2	2	98

Table 3.2: Complexity indicators of random UOV public keys. For each parameter set $(m, n, q = 31)$ we compute the maximum degree bound $D_{\max}$ using Theorem 3.2.21. The remaining columns specify how many occurrences of the respective condition were observed among 100 randomly generated public keys. The ideal J is a public key I with $\dim \mathbb{V}(I)$ linear constraints imposed (see Example 3.2.23). The values of $d_{\max}$ and i_{reg} are computed directly from the ideal J. The last column specifies how many times the Hilbert series was a polynomial, which gives a lower bound on how frequently J is regular by Lemma 3.2.20.(ii). See Appendix A.4 for details.

that because we operate in a cryptographic setting, we may assume that the system of equations p always has a solution. Also, if the system is non-homogeneous, we can homogenize it at the cost of adding another variable.

We further assume that $n = cm$ for some constant $c > 2$ and set $t = \lfloor c \rfloor - 1$. Thomae and Wolf algorithm [TW12][DPS20, Section 8.7.5] transforms p into a system of $m - t$ quadratic equations in $m - t$ variables, whose solution $a_g \in k^{m-t}$ can be lifted to a complete solution $a \in k^n$. The outline of the algorithm is as follows:

1. Map $p_1, \ldots, p_m$ to $f_1, \ldots, f_m$ of the form (3.5) below
2. Map $f_{i_1}, \ldots, f_{i_{m-t}}$ to quadratic polynomials $g_{1_1}, \ldots g_{i_{m-t}} \in k[x_{t+1}, \ldots, x_m]$
3. Find $a_g \in \mathbb{V}(g_{i_1}, \ldots, g_{i_{m-t}}) \subseteq k^{m-t}$
4. Lift $a_g \in k^{m-t}$ to a complete solution $a \in k^n$.

An implementation of this algorithm can be found in the appendix under `tw.sage`. We will now study each step in detail.

Step 1. Our goal is to find a linear transformation $B \in \mathrm{GL}(n,k)$ such that the polynomials $(f_1,\ldots,f_m) = p \circ B$ have the following form

$$f_i = \underbrace{\sum_{j=1}^{t} c_{i,j}x_j^2}_{F_1} + \underbrace{\sum_{j=1}^{t} x_j L_{i,j}(x_{m+1},\ldots,x_n)}_{F_2} + \underbrace{Q_i(x_{t+1},\ldots,x_n)}_{F_4}, \tag{3.5}$$

where $L_{i,j}$ are linear and Q_i are quadratic equations in the specified variables. Fix $i \in \{1,\ldots,m\}$ and let $F_i \in k^{n\times n}$ be the matrix of the quadratic form f_i corresponding to equation (3.5). Assuming that F_i is in upper triangular form, we can write it as follows

$$F_i = \begin{bmatrix} c_1 & \cdots & 0 & 0 & \cdots & 0 & \ell_{1,m+1} & \cdots & \ell_{1,n} \\ \vdots & \ddots & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & c_t & 0 & \cdots & 0 & \ell_{t,m+1} & \cdots & \ell_{t,n} \\ 0 & \cdots & 0 & q_{1,1} & \cdots & \cdots & \cdots & \cdots & q_{1,n-t} \\ \vdots & & \vdots & \vdots & \ddots & & & & \vdots \\ \vdots & & \vdots & \vdots & & \ddots & & & \vdots \\ \vdots & & \vdots & \vdots & & & \ddots & & \vdots \\ \vdots & & \vdots & \vdots & & & & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & \cdots & \cdots & \cdots & q_{n-t,n-t} \end{bmatrix}, \tag{3.6}$$

for some $c_*, \ell_*, q_* \in k$. The elements ℓ_* and q_* correspond to coefficients in $L_{i,j}$ and Q_i respectively. It is helpful to breakdown the matrix F_i into blocks as follows

$$F_i = \begin{bmatrix} F_{i_1} & F_{i_2} \\ 0 & F_{i_4} \end{bmatrix} = \begin{bmatrix} *_{t\times t} & *_{t\times(n-t)} \\ 0_{(n-t)\times t} & *_{(n-t)\times(n-t)} \end{bmatrix}.$$

The submatrix F_{i_2} can be further decomposed into

$$F_{i_2} = \begin{bmatrix} F_{i_{2,1}} & F_{i_{2,2}} \end{bmatrix} = \begin{bmatrix} 0_{t\times(m-t)} & *_{t\times(n-m)} \end{bmatrix}.$$

Using this notation, the goal of Step 1 is to sets $P_{i_{2,1}} = 0$ and diagonalizes P_{i_1} for all matrices $P_i \in k^{n\times n}$ of the quadratic forms corresponding to p_i.

Lemma 3.3.1. *There is $B \in \mathrm{GL}(n,k)$ such that $p \circ B$ produces the system of equations of the*

form (3.5). *Furthermore, we can write* $B = \prod_{d=2}^{m} B_d$ *where each* B_d *is of the following form*

$$B_d = \begin{bmatrix} 1 & \cdots & 0 & b_{1,d} & 0 & \cdots & 0 \\ \vdots & \ddots & & \vdots & & & \vdots \\ \vdots & & 1 & b_{d-1,d} & & & \vdots \\ \vdots & & & 1 & & & \vdots \\ \vdots & & & b_{d+2,d} & 1 & & \vdots \\ \vdots & & & \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & b_{n,d} & 0 & \cdots & 1 \end{bmatrix}.$$

Proof. We will show the existence of B by following $m-1$ steps $d = 2, \ldots, m$. Let $P_i^{(1)}$ denote the matrix of the quadratic form p_i. At step d, we compute new matrices

$$P_i^{(d)} = B_d^\intercal P_i^{(d-1)} B_d$$

that satisfy the following criteria for all $i = 1, \ldots, m$:

(a) Entries in column d above row $\min\{d, t+1\}$ of matrix $P_i^{(d)}$ are set to zero. This ensures that the corresponding entries match the form of F_i illustrated in (3.6).

(b) The top-left submatrix $P_i^{(d)}$ of dimension $(d-1) \times (d-1)$ is equal to the top-left submatrix of $P_i^{(d-1)}$ of the same dimension. This ensures that the relevant changes from the previous step are preserved.

Note, we are only concerned with the form of submatrices corresponding to F_{i_1} and $F_{i_{2,1}}$ from the discussion preceding the lemma, so all other entries may change.

It remains to show that the above criteria hold at each step. Let $d \in \{2, \ldots, m\}$ be arbitrary.

(a) Consider the decomposition of the matrix B_d into blocks of the following dimensions

$$B_d = \begin{bmatrix} B_{d_1} & 0 \\ B_{d_3} & 1 \end{bmatrix} = \begin{bmatrix} *_{m \times m} & 0_{m \times (n-m)} \\ *_{(n-m) \times m} & 1_{(n-m) \times (n-m)} \end{bmatrix}.$$

For each $i \in \{1, \ldots, m\}$, the composition of the quadratic form $P_i^{(d-1)}$ with the linear transformation B_d is given by

$$\begin{aligned} B_d^\intercal P_i^{(d-1)} B_d &= \begin{bmatrix} B_{d_1}^\intercal & B_{d_3}^\intercal \\ 0 & 1 \end{bmatrix} \begin{bmatrix} P_{i_1}^{(d-1)} & P_{i_2}^{(d-1)} \\ 0 & P_{i_3}^{(d-1)} \end{bmatrix} \begin{bmatrix} B_{d_1} & 0 \\ B_{d_3} & 1 \end{bmatrix} \\ &= \begin{bmatrix} B_{d_1}^\intercal P_{i_1}^{(d-1)} B_{d_1} + B_{d_1}^\intercal P_{i_2}^{(d-1)} B_{d_3} + B_{d_3}^\intercal P_{i_4}^{(d-1)} B_{d_3} & * \\ * & * \end{bmatrix}. \end{aligned} \tag{3.7}$$

Recall that in our setting we have $1 \leq t < m < n$. When $d < t$, we want to set the d-th column above the diagonal of $p_i \circ B_d$ to zero. This corresponds to diagonalizing F_{i_1} from the discussion above. When $t \leq d < m$ we want to set the first t elements in the d-th column of $p_i \circ B_d$ to zero, which corresponds to setting $F_{i_{2,1}} = 0$.

We setup a system of equations in variables b_* by setting the corresponding elements in column d of (3.7) to zero. Due to the fact that B_d contains 1 at position (d, d), all equations are linear. We thus obtain a system of (up to) mt linear equations in $n - 1$ variables. Since

$$mt = m(\lfloor c \rfloor - 1) < mc - m < n - 1, \tag{3.8}$$

the system has a non-trivial solution.

(b) If we split the matrix B_d into blocks of the following dimensions, we will have an identity matrix in the top-left

$$B_d = \begin{bmatrix} 1 & \tilde{B}_{d_2} \\ 0 & \tilde{B}_{d_4} \end{bmatrix} = \begin{bmatrix} 1_{(d-1)\times(d-1)} & *_{(d-1)\times(n-d+1)} \\ 0_{(n-d+1)\times(d-1)} & *_{(n-d+1)\times(n-d+1)} \end{bmatrix}.$$

Then the composition of quadratic form $P_i^{(d-1)}$ with the linear transformation B_d in terms of the new block dimensions is

$$B_d^\intercal P_i^{(d-1)} B_d = \begin{bmatrix} 1 & 0 \\ \tilde{B}_{d_2}^\intercal & \tilde{B}_{d_4}^\intercal \end{bmatrix} \begin{bmatrix} \tilde{P}_{i_1}^{(d-1)} & \tilde{P}_{i_2}^{(d-1)} \\ 0 & \tilde{P}_{i_3}^{(d-1)} \end{bmatrix} \begin{bmatrix} 1 & \tilde{B}_{d_2} \\ 0 & \tilde{B}_{d_4} \end{bmatrix} = \begin{bmatrix} \tilde{P}_{i_1}^{(d-1)} & * \\ * & * \end{bmatrix}.$$

That is, the top-left submatrix of $P_i^{(d-1)}$ of dimensions $(d-1) \times (d-1)$ is preserved by the composition. Therefore, we can transform P_i into the desired form by computing each B_d separately for $d = 2, \ldots, m$.

We have seen that each B_d can be obtained by solving a system of linear equations, which, combined with the discussion in the previous paragraph, states that

$$B_m^\intercal \cdots B_2^\intercal P_i B_2 \cdots B_m$$

has the form of equation (3.5). Since each $B_d \in \mathrm{GL}(n, k)$, the product $\prod_{j=1}^m B_d = B$ is also in $\mathrm{GL}(n, k)$. □

Step 2. Now that the transformation B is found, we apply it to p and obtain the equations $f_1, \ldots, f_m$ in the form of (3.5). The goal of this step is to map a subset of polynomials $f \in k[x_1, \ldots, x_n]^m$ into a system of equations $g \in k[x_{t+1}, \ldots, x_m]^{m-t}$.

Fix an arbitrary[4]

$$a_\ell \in \mathbb{V}(\{L_{i,j}(x_{m+1}, \ldots, x_n) \mid 1 \leq i \leq m, 1 \leq j \leq t\}).$$

The variety above is produced by mt linear equations in $n - m$ variables each. By (3.8), $mt < n - m$, so there is a non-trivial solution a_ℓ.

Now $a_\ell \in k^{n-m}$ and corresponds to the values of $x_{m+1}, \ldots, x_n$ that annihilate the terms containing $L_{i,j}$ in equation (3.5). For $i = 1, \ldots, m$ we set

$$g_i = f_i(a_\ell) = \sum_{j=i}^{t} c_{i,j} x_j^2 + G_i(x_{t+1}, \ldots, x_m), \tag{3.9}$$

where G_i is a polynomial in the specified variables. Note that G_i and g_i are no longer guaranteed to be homogeneous.

Let $g_{i_1}, \ldots, g_{i_t}$ be t arbitrarily polynomials from $g_1, \ldots, g_m$. Fix a graded monomial order and bring the Macaulay matrix $\mathbb{M}(g_{i_1}, \ldots, g_{i_t})$ into row-reduced echelon form. For illustration below, we assume that all squares x_i^2 appear before any combination $x_i x_j$ for $i \neq j$, but in actuality any graded order will work since we do not have any combinations $x_i x_j$ for $1 \leq i < j \leq t$ in equations (3.9). We can visualize the resulting echelon form as follows

$$H = \begin{bmatrix} 1 & \cdots & 0 & r_{1,t+1} & \cdots & r_{1,n} \\ \vdots & \ddots & \vdots & \vdots & & \vdots \\ 0 & \cdots & 1 & r_{t,t+1} & \cdots & r_{t,n} \end{bmatrix}.$$

By reading the polynomials off the rows of H we obtain t substitutions

$$\begin{aligned} x_1^2 &= S_1(x_{t+1}, \ldots, x_m), \\ &\vdots \\ x_t^2 &= S_t(x_{t+1}, \ldots, x_m), \end{aligned} \tag{3.10}$$

where S_i are quadratic polynomials in the specified variables with coefficients $r_* \in k$. As i ranges over $\{1, \ldots, m\} \setminus \{i_1, \ldots, i_t\}$ we substitute $x_j^2 \to S_j$ in each polynomial g_i. As evident from the form of g_i in equation (3.9) and the set of substitutions above, the resulting polynomials g_i are in $k[x_{t+1}, \ldots, x_m]$. This produces a system of $m - t$ quadratic equations in $m - t$ variables.

[4]In what follows, we will need to compute square roots over finite fields, which are not guaranteed to exist for fields of odd characteristic (see Lemma 3.4.1). Therefore any arbitrary choice of a parameter, such as this one, can be used to our advantage to increase the chance of obtaining the square roots.

Step 3. We now solve the system of equations obtained at the previous step. If a solution exists, we set

$$a_g \in \mathbb{V}(\{g_i \mid i \in \{1, \ldots, m\} \setminus \{i_1, \ldots, i_t\}\})$$

to an arbitrary element. If there is no solution we may try to choose an alternative solution $a_\ell \in \mathbb{V}(L_{i,j})$ and repeat.

The value $a_g \in k^{m-t}$ is a partial solution of the original system of equations p corresponding to the variables $x_{t+1}, \ldots, x_m$. Note, this is the most resource intensive step of the algorithm.

Step 4. It remains to lift the partial solutions from the previous steps to a complete solution $a \in k^n$ of $p(a) = 0$. Recall, we have $a_\ell \in k^{n-m}$ and $a_g \in k^{m-t}$ both of which combined cover $x_{t+1}, \ldots, x_n$, so we are only missing a solution for $x_1, \ldots, x_t$.

We can obtain a solution $a_\surd = (a_1, \ldots, a_t)$ from the equations (3.10), by computing $a_j = \sqrt{S_j(a_g)}$ for all $j = 1, \ldots, t$. When k is of odd characteristic, by Lemma 3.4.1, the solution exists with the probability $\approx \frac{1}{2^t}$. Despite this setback, recall that there were several places where we made an arbitrary choice:

- Selection of $a_\ell \in \mathbb{V}(L_{i,j})$ at Step 2
- Selection of $i_1, \ldots, i_t$ at Step 2
- Selection of $a_g \in \mathbb{V}(g_{j_1}, \ldots, g_{j_{m-t}})$ at Step 3

By making a different selection and re-executing the algorithm approximately $\frac{1}{2^t}$ times we can increase the likelihood of obtaining the square roots $a_\surd$. Then, the complete solution is given by $a = B^{-1}(a_\surd, a_g, a_\ell)$.

3.4 Relinearization Techniques and XL $(n < m)$

In this section we assume that $n < m$ and that we are given a quadratic system $p \in k[x_1, \ldots, x_n]^m$ and some $b = (b_1, \ldots, b_m) \in k^m$. Our goal is to find $a = (a_1, \ldots, a_n) \in k^n$ such that $p(a) = b$. Recall that we operate in a cryptographic setting, where we may assume that the system of equations p is consistent.

Suppose that $m = |M_{\leq 2}(x, n)| - 1$ and that the polynomials in p have no constant coefficients. If there are constant coefficients, we move them into b. For each monomial in $M_{\leq 2} \setminus \{1\}$ we introduce a variable y_i and substitute the corresponding monomials in p with variables y_i. This produces a system of linear equations $p' \in k[y_1, \ldots, y_m]^m$. Observe that if we fix a monomial order, then the Macaulay matrix $\mathbb{M}(p)$ corresponds to the coefficient matrix of p'. Furthermore, there is a bijection between y_i and monomials $x^\alpha \in M_{\leq 2} \setminus \{1\}$ defined by the monomial order. We will use $\mathbb{M}(p)$ instead of the coefficient matrix of p' going forward.

If $\mathbb{M}(p)$ is invertible, then the system of equations p' has a unique solution. We compute the inverse and obtain the solution in y_j via $a = \mathbb{M}(p)^{-1} \cdot b^\intercal$. From this, we can easily determine the solution to the quadratic system p in x_i by reading it off the elements of a at positions corresponding to monomials x_i.

If $\mathbb{M}(p)$ is singular, we eliminate all linearly dependent equations and obtain $\mathbb{M}(q)$ of dimensions $r \times m$ for some $r < m$. Next, we compute the right inverse $\mathbb{M}(q)^{-1} \in k^{m \times r}$ and obtain a solution in y_j via $a' = \mathbb{M}(q)^{-1} \cdot (b_1, \ldots, b_r)^\intercal$ in k^m. However, the solution a' might not correspond to a viable solution in x_i. We may enumerate the remaining solutions via $a' + w$ as w ranges over the kernel of $\mathbb{M}(q)$, but it could be too inefficient. There could be as many as q^{m-r} solutions by Rank-Nullity theorem, where q is the order of k. The relinearization technique, discussed below, introduces additional constraints in the form of equation (3.11) to reduce the search space. Before we discuss it, we shall briefly review the case of homogeneous polynomials.

Let the polynomials in p be homogeneous of degree two. Then the Macaulay matrix

$$\mathbb{M}(p) = \begin{bmatrix} \mathrm{MC}(p_1, x_1^2) & \mathrm{MC}(p_1, x_1x_2) & \cdots & \mathrm{MC}(p_1, x_n^2) \\ \vdots & \vdots & & \vdots \\ \mathrm{MC}(p_m, x_1^2) & \mathrm{MC}(p_m, x_1x_2) & \cdots & \mathrm{MC}(p_m, x_n^2) \end{bmatrix}$$

is in $k^{|M_2| \times |M_2|}$. Assuming $\mathbb{M}(p)$ is invertible, the unique solution in y_j is obtained as above. However, in order to map the solution from y_j back to x_i we will need to take square roots in k. This is because variables y_j correspond to monomials in x^α of degree two and so we would need to solve $x_i^2 = a_i$ for some $a_i \in k$.

Lemma 3.4.1. *Let k be a finite field and let $a \in k$ be randomly chosen using a uniform distribution. If the characteristic of k is even, then there is $r \in k$ such that $r^2 = a$. For k of odd characteristic, such element r exists with probability $\approx \frac{1}{2}$.*

Proof. If the order of k is 2^t, we can choose $r = a^{2^{t-1}}$ to obtain $r^2 = (a^{2^{t-1}})^2 = a^{2^t} = a$.

If the order of k is a power of an odd prime, then the squares of the group $k^\times$ form a subgroup of index two [Ser73, Theorem I.3.4]. Hence, the probability is approximately $1/2$. □

Consequently, recovering a solution for homogeneous p over a field of odd characteristic may be problematic. Provided $\mathbb{M}(p)$ is invertible, a complete solution exists with a probability of $\approx (1/2)^n$. Nevertheless, a partial solution is likely to exist, which can be substituted into p to obtain a new system of equations p' with fewer variables. Then the linearization process can be repeated for a smaller system p'.

For the rest of this section we assume that k is of characteristic two, p is homogeneous, and that $\mathbb{M}(p)$ is of full rank. So far, the most efficient way to obtain a solution involved having

$m = |M_2|$ and computing the inverse of $\mathbb{M}(p)$. However, we can obtain a similarly efficient algorithm when $m \approx \frac{1}{10}|M_2|$. This result is due to Aviad Kipnis and Adi Shamir [KS99] who introduced the algorithm known as the relinearization technique.

We start with the linearization of $p = (p_1, \ldots, p_m)$ via the Macaulay matrix $\mathbb{M}(p)$ as described above. This yields a system of linear equations q in variables $y_{i,j}$ corresponding to $x_i x_j \in M_2$

$$q = (q_1, \ldots, q_m) \in k[y_{1,1}, \ldots, y_{i,j}, \ldots, y_{n,n}]^m, \quad \text{where } 1 \leq i \leq j \leq n.$$

Due to the change of variables via linearization, we obtain a number of equations in $y_{i,j}$, which always hold in the ring $k[x_1, \ldots, x_n]$

$$y_{a,b}y_{c,d} = y_{a,c}y_{b,d} = y_{a,d}y_{b,c} \iff (x_a x_b)(x_c x_d) = (x_a x_c)(x_b x_d) = (x_a x_d)(x_b x_c). \quad (3.11)$$

Lemma 3.4.2. *Let E be the set of equations* (3.11) *then $|E| = 2\binom{n}{4}$ and all equations in E are linearly independent.*

Proof. [KS99, Section 5.2]. □

If we consider the system of equations $q' = q \cup E$, we have more equations but we also changed the system from linear to quadratic. The idea of the relinearization technique is precisely as the name suggests. We linearize the system again by considering the Macaulay matrix $\mathbb{M}(q')$. Since the system q is linear in $y_{i,j}$ and E is quadratic, we can be certain that the equations in E do not appear in q and vice-versa. Therefore, the relinearization yields a system of equations

$$r = (r_1, \ldots, r_{m+|E|}) \in k[z_1, \ldots, z_t]^{m+|E|},$$

where the number of variables z_i is given by $t = |M_2(z, s)|$ for $s = |M_2(x, n)|$. If the number of equations $m + |E| \geq t$ we can solve the system using Gaussian elimination in polynomial time using the same approach as above.

Lemma 3.4.3 (Relinearization Technque)**.** *Let* $\mathrm{char}(k) = 2$*, $b \in k^m$, $p \in k[x_1, \ldots, x_n]^m$. Suppose p is homogeneous of degree two, $m \approx \frac{1}{10}|M_2(x, n)|$ and $\mathbb{M}(p)$ is of full rank. Then the solution $a \in k^n$ satisfying $p(a) = b$ can be found in $\approx O(m^\omega)$ operations in k.*

Proof. Correctness of the algorithm is shown in [KS99]. Since the computation of the matrix inverse is the dominant operation, the time complexity follows by Lemma 3.2.17. □

Extended Linearization It is possible to continue the relinearization of a system by adding more equations of higher degrees. For example, we can add

$$y_{a,b}y_{c,d}y_{e,f} = y_{a,c}y_{b,d}y_{e,f} \iff (x_ax_b)(x_cx_d)(x_ex_f) = (x_ax_c)(x_bx_d)(x_ex_f).$$

However, not all of the placements of parenthesis in the equations above yield linearly independent equations. Due to this fact, the complexity analysis of the relinearization technique for higher degrees becomes difficult [CKPS00, TW10]. Nevertheless, the ideas of the relinearization technique have been extended into an algorithm called eXtended Linearization or XL for short [CKPS00].

Lemma 3.4.4 (XL)**.** *Let $p \in k[x_1, \ldots, x_n]^m$ be homogeneous of degree two and let $b \in k^m$. Then the algorithm below terminates for some $D \in \mathbb{N}$ and yields a solution $a \in k^n$ such that $p(a) = b$. The complexity of the algorithm is $O\left(\binom{n+D+2}{D+2}^{\omega}\right)$.*

For $i = 1, \ldots, n$. Fix a monomial order such that monomials x_i^d are strictly greater than any other monomials. Set $D = 1$ and proceed as follows

1. *Set $I_D = \bigcup_{i=0}^{D}\{x^\alpha f_j \mid 1 \leq j \leq m, \alpha \in M_i(x, n)\}$*
2. *If $\text{rank}(\mathbb{M}(I_D)) < |M_D(x, n)|$, set $D = D + 1$ and go back to step 1*
3. *Bring $\mathbb{M}(I_D)$ into row-reduced echelon form*
4. *Solve the univariate polynomial in x_i specified in the first D rows of $\mathbb{M}(I_D)$*
5. *Substitute solution x_i into p and proceed to next i*

Proof. Algorithm and initial analysis [CKPS00]; complexity analysis and a proof that relinearization is a special case of XL [TW10]. □

Extended linearization and in particular its variant known as Wiedemann XL [Moh11] is among the most popular polynomial system solving algorithms in the cryptographic community. However, as demonstrated by Gwénolé Ars et al. [AFI+04], XL algorithms are equivalent to Gröbner basis algorithms. The popularity of Wiedemann XL is motivated by low memory requirements, which makes practical applications more convenient.

Chapter 4

Polynomial Equivalence Problem

The goal of this chapter is to study the Polynomial Equivalence Problem (PEP) for the Unbalanced Oil and Vinegar (UOV) signature scheme. Informally, the goal of the PEP is to find a private key from the specified public key. Once the private key is found it can be used for signing arbitrary messages, so the difficulty of solving PEP has a direct impact on the security of UOV. We will now state this problem formally.

Definition 4.0.1 (Polynomial Equivalence Problem)**.** Let $p \in R^m$ be a public key of Unbalanced Oil and Vinegar signature scheme. We say that $T \in \mathrm{GL}(n, k)$ is a solution to the polynomial equivalence problem if $p \circ T$ is a central map.

Observe that a solution to PEP produces an equivalent private key $(p \circ T, T^{-1})$, which can be used to sign messages in the same way as the original private key. That is, the alternative central map $f = p \circ T$ is easily invertible and so is T. The computation of the alternative central map f is efficient since $p \circ T$ is obtained by a linear change of variables $p(T(x))$ for $x = (x_1, \ldots, x_n) \in k[x_1, \ldots, x_n]^m$.

The definition of the PEP above is closely related to the Isomorphism of Polynomials (IP) problem proposed by Jacques Patarin along with a multivariate quadratic system called Hidden Field Equations [Pat96]. The difference between PEP and IP is that the former assumes the central map to be a part of the private key, while the latter assumes that the central map is known to the attacker. In case of UOV the central map is private so the IP problem is not applicable[1].

[1] A more general version of PEP, called Extended Isomorphism of Polynomials (EIP), is defined by Ding et al. [DPS20, Definition 2.16]. The difference with Definition 4.0.1 is that EIP has two linear transformations while PEP uses one (UOV only has one). To avoid confusion with IP, whose central maps are public, we will use the name PEP.

There are three known solutions of the polynomial equivalence problem, which are better than direct approaches for certain parameter choices. The solutions are: Kipnis-Shamir attack [KS98], reconciliation attack [DYC+08], and intersection attack [Beu21]. We will discuss all three attacks below.

4.1 Solving PEP Using O_x

In order to make the central map of UOV efficiently invertible, the polynomials are required to be linear in the oil variables. However, in the public key, this relationship is concealed by the secret linear transformation. If we consider the inputs to the central map as elements of the vector space k^n, we can think of each vector in terms of its oil and vinegar components

$$(\underbrace{u_1, \ldots\ldots, u_{n-m}}_{\text{Vinegar}}, \underbrace{u_{n-m+1}, \ldots, u_n}_{\text{Oil}}) \in k^n.$$

These components form so-called oil and vinegar subspaces, which are formally defined below. The inputs to the public key are also vectors of k^n, but we can no longer discern the difference between oil and vinegar components since they were mixed by the secret linear transformation

$$(\underbrace{v_1, \ldots\ldots, v_{n-m}, v_{n-m+1}, \ldots, v_n}_{\text{Mixed Oil and Vinegar}}) \in k^n.$$

However, since the linear transformation must be invertible, it is onto and so there is still an oil subspace of dimension m hiding in the inputs to the public key. Kipnis-Shamir attack (Section 4.2) shows how to determine a basis of this subspace for $n = 2m$. In this section we will focus on a more general result – how to obtain a solution to PEP from a basis of O_x.

Definition 4.1.1 (Oil Space)**.** Let $A \in \mathrm{GL}(n, k)$ be a secret linear transformation of the private key of UOV. We will denote the public variables by x_i, the secret variables by y_i, and define the oil and vinegar subspaces, O_* and V_* respectively, as follows:

$$\begin{array}{lll} y = Ax = (y_1, \ldots, y_n), & O_y = \mathrm{span}\{e_{n-m+1}, \ldots, e_n\}, & V_y = \mathrm{span}\{e_1, \ldots, e_{n-m}\}, \\ x = (x_1, \ldots, x_n), & O_x = A^{-1}O_y, & V_x = A^{-1}V_y, \end{array}$$

where $e_1, \ldots, e_n$ is the standard basis of k^n.

Observe that the vector space k^n can be written as a direct sum of the oil and vinegar subspaces. Since the linear map A is invertible, this relationship is preserved under the linear change of variables. In particular, the following holds

$$y = O_y \oplus V_y = A(O_x \oplus V_x),$$

$$x = O_x \oplus V_x = A^{-1}(O_y \oplus V_y).$$

Our goal is to show that a solution to PEP can be obtained by determining a basis of a vector subspace O_x. This result may seem fairly intuitive if we consider an extension of the basis from O_x to k^n. We will provide a constructive proof in Theorem 4.1.3 below to establish that basis extension is all that is required. However, there is also an interesting observation about what such extensions might look like.

Let $u_1, \ldots, u_m$ be a basis of O_x. Suppose we managed to superimpose the components (u_{ji}) of the basis u_i onto an identity matrix and obtained a linear transformation of the following form[2]

$$B = \begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & u_{11} & \cdots & u_{1m} \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & u_{m1} & \cdots & u_{mm} \end{bmatrix}.$$

Can this be a solution to PEP?

Lemma 4.1.2. *Let $p \in R^m$ be a public key such that $p \notin S^m$. Then no matrix $B \in \mathrm{GL}(n, k)$ of the following form is a solution to the polynomial equivalence problem*

$$B = \begin{bmatrix} B_1 & 0 \\ 0 & B_4 \end{bmatrix}.$$

Proof. Without loss of generality we may assume that $m = 1$. If we write the matrix of quadratic form p as

$$P = \begin{bmatrix} P_1 & P_2 \\ P_3 & P_4 \end{bmatrix},$$

then the submatrix P_4 of dimension $m \times m$ cannot be zero by Lemma 2.2.3. Suppose there is a solution B to PEP of the form above. Then

$$B^\intercal P B = \begin{bmatrix} B_1^\intercal & 0 \\ 0 & B_4^\intercal \end{bmatrix} \begin{bmatrix} P_1 & P_2 \\ P_3 & P_4 \end{bmatrix} \begin{bmatrix} B_1 & 0 \\ 0 & B_4 \end{bmatrix} = \begin{bmatrix} * & * \\ * & B_4^\intercal P_4 B_4 \end{bmatrix}.$$

Since we assume $B^\intercal P B$ to be a central map we must have $B_4^\intercal P_4 B_4 = 0$ by Lemma 2.2.3, but B_4 is invertible so we multiply by $(B_4^\intercal)^{-1}$ on the left and by B_4^{-1} on the right. Then $P_4 = 0$, which is a contradiction. □

[2] It is worth comparing matrix B with the form of the matrix (4.3) recovered by the reconciliation attack.

Theorem 4.1.3. *Let $e_1, \ldots, e_n$ be the standard basis of k^n. If $u_1, \ldots, u_m \in k^n$ is a basis of O_x, then there is $B \in \mathrm{GL}(n,k)$ such that $B \cdot e_{n-m+i} = u_i$ for all $i = 1, \ldots, m$ and B is a solution to the polynomial equivalence problem.*

Proof. Let $p = f \circ A$ be a public key. We are given the basis $u_1, \ldots, u_m \in k^n$ of the oil space O_x. First, we extend this basis to a new basis $v_1, \ldots, v_n \in k^n$ so that $v_{n-m+i} = u_i$ for all $i = 1, \ldots, m$. Note that v_i's span $k^n = V_x \oplus O_x$. Second, we define the linear transformation $B \in \mathrm{GL}(n,k)$ by $B \cdot e_i = v_i$. Observe that $B \cdot e_{n-m+i} = u_i$.

Claim 1: Subspaces O_y and V_y are invariant under $AB \in \mathrm{GL}(n,k)$. Let $w \in O_y$, then $w = \alpha_1 e_{n-m+1} + \cdots + \alpha_m e_n$ and so $B \cdot w = \alpha_1 u_1 + \cdots + \alpha_m u_m$ is in O_x. Since $O_x = A^{-1} O_y$ there is some $w' \in O_y$ such that $B \cdot w = A^{-1} \cdot w'$. Then $AB \cdot w = w' \in O_y$. The result for V_y follows by a similar argument.

Let $C = AB$. Since $k^n = V_y \oplus O_y$, $CV_y = V_y$ and $CO_y = O_y$, the matrix of C has the following form

$$C = \begin{bmatrix} C_1 & 0 \\ 0 & C_4 \end{bmatrix},$$

where $C_4 \in k^{m \times m}$.

Claim 2: The linear transformation B is a solution to PEP. We need to show that the composition $p \circ B$ is in S. Fix $i \in \{1, \ldots, m\}$ and let $P, F \in k^{n \times n}$ denote the matrices of quadratic forms p_i and f_i respectively. Since $p_i(x) = f_i(A \cdot x)$ we have $p_i(x) = x^\intercal P x = x^\intercal (A^\intercal F A) x = f_i(A \cdot x)$, and therefore $P = A^\intercal F A$. Then

$$p \circ B = B^\intercal P B = (AB)^\intercal F (AB) = C^\intercal F C.$$

The bottom-right submatrix of F is zero by Lemma 2.2.3, so the product

$$C^\intercal F C = \begin{bmatrix} C_1^\intercal & 0 \\ 0 & C_4^\intercal \end{bmatrix} \begin{bmatrix} F_1 & F_2 \\ F_3 & 0 \end{bmatrix} \begin{bmatrix} C_1 & 0 \\ 0 & C_4 \end{bmatrix} = \begin{bmatrix} C_1^\intercal F_1 C_1 & C_1^\intercal F_2 C_4 \\ C_4^\intercal F_4 C_1 & 0 \end{bmatrix}.$$

Applying Lemma 2.2.3 in the opposite direction, we obtain that $p \circ B$ is a central map and hence B is a solution to PEP. □

4.2 Kipnis-Shamir Attack

Linear transformations and quadratic forms behave differently under the linear change of variables $y = Ax$ for some $A \in \mathrm{GL}(n,k)$. If $T, Q \in k^{n \times n}$ are matrices of a linear map and a quadratic form respectively, then the former is transformed by $A^{-1} T A$ and the latter by $A^\intercal Q A$.

Aviad Kipnis and Adi Shamir observed [KS98] that if we consider products of matrices of quadratic forms $P^{-1}Q$, then they will behave similarly to linear transformations

$$(A^\intercal PA)^{-1}(A^\intercal QA) = A^{-1}P^{-1}(A^\intercal)^{-1}A^\intercal QA = A^{-1}P^{-1}QA. \tag{4.1}$$

Applied to the public keys of the Oil and Vinegar signature scheme this observation leads to an efficient procedure to recover the oil space O_x. It is precisely because of this attack that the Oil and Vinegar signature scheme, originally proposed by Jacques Patarin [Pat97], is now called Unbalanced Oil and Vinegar [KPG99] and we require $n > 2m$.

Lemma 4.2.1. *Let $n = 2m$. Suppose that $p = (p_1, \ldots, p_m) \in R^m$ is a UOV public key and $P_i, P_j \in k^{n\times n}$ are matrices of quadratic forms p_i and p_j for some $i \neq j$. If P_i and P_j are invertible, then*

$$(P_i^{-1}P_j)O_x = O_x.$$

Proof. Since p is a public key, there is $f = (f_1, \ldots, f_m) \in S^m$ and $A \in \mathrm{GL}(n, k)$ such that $p = f \circ A$. Fix $i \neq j$ in $\{1, \ldots, m\}$. Let $F_i, F_j \in k^{n\times n}$ be matrices of quadratic forms f_i and f_j. By Lemma 2.2.3, these matrices have a zero submatrix at the bottom right. Furthermore, F_i and F_j must be invertible because P_i and P_j are invertible. If $v \in O_y$, then

$$F_j v = \begin{bmatrix} F_{j_1} & F_{j_2} \\ F_{j_3} & 0 \end{bmatrix} \begin{bmatrix} 0 \\ v_2 \end{bmatrix} = \begin{bmatrix} F_{j_2}v_2 \\ 0 \end{bmatrix} \in V_y, \tag{4.2}$$

so $F_j O_y = V_y$, because F_j is onto and $\dim O_y = \dim V_y$ when $n = 2m$. Similarly, $F_i^{-1}V_y = O_y$. Therefore, $(F_i^{-1}F_j)O_y = O_y$.

If $v \in O_x$, then there is $w \in O_y$ such that $v = A^{-1}w$. Applying (4.1), we obtain

$$(P_i^{-1}P_j)v = (A^{-1}F_i^{-1}F_jA)A^{-1}w = A^{-1}w',$$

for some $w' \in O_y$. Consequently, $(P_i^{-1}P_j)O_x = O_x$. □

The lemma above suggests that the oil space could be found by considering invariant subspaces with respect to the linear transformations $P_i^{-1}P_j$, where both P_i and P_j are invertible. To aid with the search for the invariant subspace we would like to obtain more linear transformations with the desired property. This can be achieved using the following result [KS98, Theorem 7].

Theorem 4.2.2. *Let $n = 2m$ and let $p \in R^m$ be a public key. Discard all singular matrices P_ℓ corresponding to quadratic forms of the public key components p_ℓ. Let $I \subseteq k^{n\times n}$ be the closure of products $(P_i^{-1}P_j)$ with respect to addition, multiplication, and scalar multiplication. If $I \neq \emptyset$, then O_x is invariant under all elements of I. We say that O_x is a common invariant subspace under I.*

Proof. By Lemma 4.2.1, the generators $P_i^{-i}P_j \in I$ map O_x into itself. Since O_x is a subspace of k^n, linear combinations of elements of I also map O_x into itself. The product of elements of $A, B \in I$ has the same property because $(AB)v = A(Bv)$. □

Invariant subspaces of k^n with respect to a linear transformation can be found from characteristic polynomials using methods similar to obtaining the Jordan form. However, in this case, we do not need all characteristic roots to be in k. The following result serves most of our needs.

Theorem 4.2.3. *Let $T \in k^{n\times n}$ be a linear transformation whose minimal polynomial $g \in k[x]$ is factored into $g = g_1^{d_1} \cdots g_s^{d_s}$, where $g_i \in k[x]$ are irreducible over k. Then we can write*

$$k^n = V_1 \oplus \cdots \oplus V_s,$$

where each $V_i = \ker g_i(T)^{d_i}$ is invariant under T and $g_i^{d_i}$ is the minimal polynomial of a linear map induced by T on V_i.

Proof. [Her75, Theorem 6.6.1]. □

Let $n = 2m$ and $B \in I$ from Theorem 4.2.2. If g is the characteristic polynomial of B with two distinct irreducible factors g_1 and g_2 such that $\deg(g_1) = \deg(g_2) = m$. Then $\ker g_1(B)$ or $\ker g_2(B)$ must be O_x by Theorem 4.2.3. It remains to figure out which one. The following lemma provides means to distinguish vectors of O_x from those of V_x.

Lemma 4.2.4. *Let $n > m$. If $p \in R^m$ is a public key, then $p(O_x) = 0$.*

Proof. Since p is a public key, there is $A \in \mathrm{GL}(n, k)$ and $f \in S^m$ such that $p = f \circ A$. Fix $i \in \{1, \ldots, m\}$ and consider the matrices $P_i, F_i \in k^{n\times n}$ of quadratic forms p_i and f_i respectively.

Recall that $y = Ax$ and $O_y = \mathrm{span}\{e_{n-m+1}, \ldots, e_n\}$. Then, given $v \in O_y$, we have

$$f_i(v) = v^\intercal F_i v = \begin{bmatrix}0 & v_2^\intercal\end{bmatrix} \begin{bmatrix}F_{i_1} & F_{i_2} \\ F_{i_3} & 0\end{bmatrix} \begin{bmatrix}0 \\ v_2\end{bmatrix} = \begin{bmatrix}0 & v_2^\intercal\end{bmatrix} \begin{bmatrix}F_{i_2}v_2 \\ 0\end{bmatrix} = 0.$$

Let $u \in O_x$ be arbitrary. Since $O_x = A^{-1}O_y$ there is some $w \in O_y$ such that $u = A^{-1}w$. Therefore,

$$p_i(u) = u^\intercal P_i u = (A^{-1}w)^\intercal (A^\intercal F_i A)(A^{-1}w) = w^\intercal F_i w = 0.$$ □

Combining the results above leads to the original Kipnis-Shamir attack on balanced Oil and Vinegar with $n = 2m$. Let $p \in R^m$ be a public key and suppose the set I from Theorem 4.2.2 is not empty.

1. Compute a random element $B \in I$.

2. Compute the characteristic polynomial g of B.
3. If g does not factor into g_1 and g_2 of degree m, then go back to Step 1.
4. Find bases of $g_1(B) \cdot k^n$ and of $g_2(B) \cdot k^n$.
5. Apply Lemma 4.2.4 to determine which basis corresponds to O_x.
6. Extend the basis of O_x to k^n. By Theorem 4.1.3 we obtain a solution to PEP.

If an element B satisfying the condition of Step 3 is found on the first attempt, the complexity of the algorithm is dominated by Steps 2 and 3. A simple way to compute the characteristic polynomial $g = \det(x - A)$ in $k[x]$ is to perform Gaussian elimination and to multiply the diagonal. This can be done in time $O(n^3)$ or less [KG85]. To factor a polynomial g of degree n over k, we can apply Berlekamp's algorithm [Ber67][DPS20, Theorem 8.10] using $O\left(n^\omega + n(\log n \log(\log n)) \log q\right)$ operations in k. We estimate the complexity of a single step of the Kipnis-Shamir algorithm to be approximately $O(n^3)$ operations.

The authors of the attack argued that such an element B can be found with sufficiently high probability, but the details were left unspecified at the time [KS98]. This has been addressed by Aviad Kipnis, Jacques Patarin, and Louis Goubin in the same article that also introduced Unbalanced Oil and Vinegar to the world [KPG99].

Lemma 4.2.5. *Let $n \geq 2m$ and suppose $p = (p_1, \ldots, p_m) \in R^m$ is a public key. If P_i and P_j are invertible matrices in $k^{n\times n}$ corresponding to quadratic forms p_i and p_j, then*

$$\dim\left(P_i O_x \cap P_j O_x\right) \geq 3m - n.$$

Proof. Let U, W be subspaces of a finite dimensional vector space V and denote by $U + W$ the subspace of V defined by $\{u + w \mid u \in U, w \in W\}$. By [Her75, Corollary to Lemma 4.2.6], the following holds

$$\dim(U \cap W) = \dim U + \dim W - \dim(U + W).$$

Let $f \in S^m$ and $A \in \mathrm{GL}(n, k)$ be such that $p = f \circ A$. Denote the matrices of central maps f_i, f_j by F_i and F_j, respectively. Note that (4.2) holds even when $n > 2m$, so $FO_y \subseteq V_y$ for any matrix of the central map F. Because $\dim O_y = m$, $\dim V_y = n - m$, and F_i, F_j are invertible we obtain

$$\begin{aligned} \dim F_j O_y &= m, & \dim F_i^{-1} F_j O_y &= m, & \dim F_i^{-1} V_y &= n - m, \\ F_j O_y &\subseteq V_y, & F_i^{-1} F_j O_y &\subseteq F_i^{-1} V_y, & F_i^{-1} V_y &\supseteq O_y. \end{aligned}$$

Let $X = F_i^{-1} F_j O_y$. Observe that the vector space $F_i^{-1} V_y$ contains subspaces X and O_y. Since $X + O_y$ is also a subspace of $F_i^{-1} V_y$, we must have $\dim(X + O_y) \leq n - m$. Applying the result from the first paragraph of this proof yields

$$\dim(X \cap O_y) = \dim X + \dim O_y - \dim(X + O_y) \geq m + m - (n - m) = 3m - n.$$

We can now consider the intersection of P_iO_x and P_jO_y. By Definition 4.1.1 and Equation (4.1), we have

$$P_i^{-1}P_jO_x \cap O_x = A^{-1}F_i^{-1}F_jO_y \cap A^{-1}O_y = A^{-1}(X \cap O_y),$$

therefore $\dim(P_iO_x \cap P_jO_x) \geq 3m - n$. □

Theorem 4.2.6 (Kipnis-Patarin-Goubin)**.** *Let $n > 2m$ and suppose $p \in R^m$ is a public key such that some of the matrices P_ℓ are invertible. A solution $B \in k^{n\times n}$ to the polynomial equivalence problem can be found in approximately*

$$O(q^{-(2m-n+1)}n^4)$$

operations in the field k of order q.

Proof. Let $P_i, P_j \in k^{n\times n}$ be invertible matrices of the public key. If $w \neq 0$ in O_x is an eigenvector of $P_i^{-1}P_j$, then the subspace $W = \{\alpha w \in k^n \mid \alpha \in k\}$ is invariant under $P_i^{-1}P_j$. Therefore, we can estimate the probability that $P_i^{-1}P_j$ has a nontrivial invariant subspace that is also a subspace of O_x using the expected number of eigenvectors in O_x.

Claim: The probability that the linear map $P_i^{-1}P_j$ has a nontrivial invariant subspace that is also a subspace of O_x is approximately q^{2m-n+1}. Let $w \neq 0$ in O_x. The probability that P_jw is in $P_iO_x \cap P_jO_x$ is determined by the number of elements in the intersection divided by the number of elements in the image P_jO_x. By Lemma 4.2.5, this is

$$\frac{|P_iO_x \cap P_jO_x|}{|P_jO_x|} \geq q^{3m-n}q^{-m}.$$

If $P_jw \in P_iO_x \cap P_jO_x$, the probability that $P_i^{-1}P_jw = \lambda w$ for some $\lambda \neq 0$ in k is

$$\frac{q-1}{|O_x|} \approx q^{1-m},$$

because $P_i^{-1}P_jw$ must be in O_x. There are $q^m - 1$ nonzero $w \in O_x$, so the expected number of eigenvectors in O_x is approximately

$$q^{3m-n}q^{-m}q^{1-m}q^m = q^{2m-n+1}.$$

We can now adjust the algorithm above to work with invariant subspaces of dimension 1. Instead of choosing $B \in I$ at Step 1, we set $B = L_1^{-1}L_2$, where L_1 and L_2 are random linear combinations of invertible matrices P_ℓ. At Step 3, we no longer require the characteristic polynomial to factor into two polynomials of degree m. Any factorization will work. Then, to find m eigenvectors in O_x we estimate that the algorithm needs to be run approximately $q^{-(2m-n+1)}n$ times with each execution taking $O(n^3)$. □

4.3 Reconciliation Attack

The problem statement of the reconciliation attack matches exactly the statement of the polynomial equivalence problem. That is, given $p \in R^m$, the reconciliation attack searches for a $B \in \mathrm{GL}(n,k)$ such that $p \circ B$ has the form of a central map. Instead of looking for a generic transformation B, Jintai Ding et al. [DYC+08][DPS20, Section 5.3] observed that there are secret maps B of a particularly convenient form specified in (4.3). While such solutions are not guaranteed to exist for all public keys p, the probability that there is such a solution for an arbitrarily chosen public key is fairly high[3].

Theorem 4.3.1. *Let $p \in R^m$ be a public key and denote by q the order of the field k. With probability strictly greater than $1 - \frac{1}{q-1}$, there is $B \in \mathrm{GL}(n,k)$ of the form*

$$B = \begin{bmatrix} 1 & B_2 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1_{(n-m)\times(n-m)} & *_{(n-m)\times m} \\ 0_{m\times(n-m)} & 1_{m\times m} \end{bmatrix}, \tag{4.3}$$

such that $p \circ B$ is a central map. Furthermore, B^{-1} has the same form as B.

Proof. The argument below is independent of m, so for simplicity we assume that the public key has only one component (i.e., $m = 1$). Since p is a public key, there is $f \in S$ and $A \in \mathrm{GL}(n,k)$ such that $p = f \circ A$.

Write the secret linear transformation A in terms of submatrices of the respective dimensions

$$A = \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix}.$$

If the submatrix A_1 is invertible, we can write $A = CB$ by setting $B_2 = A_1^{-1}A_2$ and choosing the matrix C as follows

$$C = \begin{bmatrix} A_1 & 0 \\ A_3 & A_4 - A_3B_2 \end{bmatrix}.$$

Then

$$CB = \begin{bmatrix} A_1 & 0 \\ A_3 & A_4 - A_3B_2 \end{bmatrix} \begin{bmatrix} 1 & B_2 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} A_1 & A_1B_2 \\ A_3 & A_3B_2 + A_4 - A_3B_2 \end{bmatrix} = A.$$

Since C has a zero submatrix in the top-right, by Theorem 2.3.7, C is a sustaining transformation. In other words, $C^\intercal FC$ is some central map F'. Denote by F and P the matrices of quadratic forms corresponding to f and p respectively. Then

$$P = A^\intercal FA = B^\intercal(C^\intercal FC)B = B^\intercal F'B,$$

[3]We only give a lower bound but the probability is equivalent to a randomly chosen matrix of dimensions $(n-m)\times(n-m)$ being invertible.

so B is a solution to PEP.

The existence of B stems from the fact that the submatrix A_1 is invertible. Therefore, the probability of having a solution to PEP of the form (4.3) is strictly greater than $1 - \frac{1}{q-1}$ by Corollary 2.1.3.

To see that B is invertible and its inverse has precisely the same form as B, observe that

$$1 = \begin{bmatrix} 1 & B_2 - B_2 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & B_2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -B_2 \\ 0 & 1 \end{bmatrix} = BB^{-1} = B^{-1}B.$$

□

If $p = f \circ A \in R^m$ is a public key and the submatrix A_1 of A is invertible, then, by Theorem 4.3.1, there is $B^{-1} \in \mathrm{GL}(n, k)$ of the form (4.3) such that $p = f' \circ B^{-1}$ for some central map $f' \in S^m$. That is, (f', B) is an equivalent key. Let P_i, F_i' be the matrices of quadratic forms corresponding to components p_i and f_i' respectively. We do not know F_i', but we know that the bottom-right submatrix of any central map must be zero by Lemma 2.2.3. Consequently, to find B we can setup m equations of the form below by comparing the bottom-right submatrix of $p_i \circ B$ to zero

$$p_i \circ B = B^{\intercal} P_i B = F_i' = \begin{bmatrix} * & * \\ * & 0_{m \times m} \end{bmatrix}. \tag{4.4}$$

For each $i = 1, \ldots, m$ we would get m^2 equations, so there are m^3 equations in total. The number of variables is determined by the size of the submatrix B_2, which is strictly greater than m^2 when $n > 2m$.

Instead of searching for an equivalent key $B \in \mathrm{GL}(n, k)$ by considering all unknown elements of the submatrix B_2 at once, it is possible to split the procedure into m steps. Observe that if we have two matrices $B', B'' \in \mathrm{GL}(n, k)$ of the following form

$$\begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & b_{1,j} & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & b_{n-m,j} & \cdots & 0 \\ 0 & \cdots & 0 & 1 & \cdots & 0 & \cdots & 0 \\ \vdots & & \vdots & \vdots & \ddots & & & \vdots \\ \vdots & & \vdots & \vdots & & \ddots & & \vdots \\ \vdots & & \vdots & \vdots & & & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}, \tag{4.5}$$

then their product is

$$B'B'' = \begin{bmatrix} 1 & B_2' \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & B_2'' \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & B_2'' + B_2' \\ 0 & 1 \end{bmatrix}.$$

Therefore, we can write B as a product of m matrices of the form (4.5) whose nonzero columns $b_{*,j}$ are precisely the nonzero columns of B_2. Specifically, we write $B = B_{(1)}B_{(2)}\cdots B_{(m)}$ and substitute this into equation (4.4) to obtain

$$F_i' = B_{(m)}^{\intercal}\cdots\underbrace{B_{(2)}^{\intercal}\underbrace{\underbrace{B_{(1)}^{\intercal}P_iB_{(1)}}_{P_i^{(1)}}B_{(2)}}_{P_i^{(2)}}\cdots B_{(m)}}_{P_i^{(m)}=F_i'}.$$

Then, at each step $d = 1, \ldots, m$, we seek a matrix $B_{(d)}$ such that the bottom-right submatrix of dimension $d \times d$ equal to zero. This produces d equations in $n - m$ unknowns $b_{*,n-d+1}$, depicted in (4.5), for each component of the public key i.

To recapitulate, we split the procedure into m steps, denoted by d, and at each step we solve the following system for all $i = 1, \ldots, m$ simultaneously

$$P_i^{(d)} = B_{(d)}^{\intercal}P_i^{(d-1)}B_{(d)} = \begin{bmatrix} * & * \\ * & 0_{d\times d} \end{bmatrix}, \tag{4.6}$$

where $P_i^{(0)} = P_i$ and $P_i^{(m)} = F_i'$. The types of equations obtained at each step are outlined in the following result.

Lemma 4.3.2. *Let P be a symmetric matrix in $k^{n\times n}$ corresponding to the public key. For a fixed $d \in \{1, \ldots, m\}$ the equation* (4.6) *produces one quadratic equation in $n - m$ variables and up to $d - 1$ distinct linear equations.*

Proof. Denote P by $P^{(0)}$ and let $d \in \{1, \ldots, m\}$. We are interested in the bottom right $m \times m$-submatrix of $P^{(d)}$

$$B_{(d)}^{\intercal}P^{(d-1)}B_{(d)} = \begin{bmatrix} 1 & 0 \\ B_2^{\intercal} & 1 \end{bmatrix} \begin{bmatrix} P_1 & P_2 \\ P_3 & P_4 \end{bmatrix} \begin{bmatrix} 1 & B_2 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} * & * \\ * & B_2^{\intercal}P_1B_2 + B_2^{\intercal}P_2 + P_3B_2 + P_4 \end{bmatrix},$$

where

$$P_4 = \begin{bmatrix} * & * \\ * & 0_{(d-1)\times(d-1)} \end{bmatrix} \quad \text{and} \quad B_2 = \begin{bmatrix} 0 & \cdots & b_{1,n-d+1} & \cdots & 0 \\ \vdots & & \vdots & & \vdots \\ 0 & \cdots & b_{n-m,n-d+1} & \cdots & 0 \end{bmatrix}.$$

The first term $B_2^\intercal P_1 B_2 = (\alpha_{ij})$ is the only one contributing quadratic equations, which, due to the structure of B_2, has all elements except for $\alpha_{n-d+1,n-d+1}$ set to zero. Therefore, we only have one quadratic equation.

Since P is symmetric, we have $P_2 = P_3^\intercal$ and thus $B_2^\intercal P_2 = (P_3 B_2)^\intercal$. Consequently, the equations in the column $n-d+1$ and row $n-d+1$ are the same, which yields $d-1$ linear equations. Due to zeros in B_2 and P_4, the bottom right submatrix of $P^{(d)}$ of dimension $(d-1)\times(d-1)$ is zero, so all equations are accounted for. □

The complexity of the attack is dominated by step $d = 1$ because there are only quadratic equations and no linear equations by Lemma 4.3.2. The system of equations (4.6) produces m quadratic equations $g_1, \ldots, g_m$ in $k[b_1, \ldots, b_{n-m}]$. Assuming g_i behave as a random system and there is a solution B of the required form, we can impose $n-2m$ linear constraints as in Example 3.2.23 to obtain a system of $n-m$ equations in $n-m$ variables. If the system is regular and if it has a solution, we can estimate the complexity of the attack using Lemma 3.2.22.

Lemma 4.3.3. *Let $n \geq 2m$, $p \in R^m$ and suppose there is $B \in \mathrm{GL}(n,k)$ of the form* (4.3) *such that $p \circ B \in S^m$. Let $g_1, \ldots, g_m$ and $\ell_1, \ldots, \ell_{n-2m}$ in $k[b_1, \ldots, b_{n-m}]$ be such that g_i correspond to quadratic equations of* (4.6) *for $d = 1$ and ℓ_i are random linear equations. If $\dim \mathbb{V}(g_1, \ldots, g_m) = n - 2m$ and the system of equations $J = \langle g_1, \ldots, g_m, \ell_1, \ldots, \ell_{n-2m}\rangle$ is regular with $\dim \mathbb{V}(J) = 0$ and $\mathbb{V}(J) \neq \emptyset$, then B can be found in time*

$$O\left(\binom{n-m+i_{\mathrm{reg}}}{n-m}^{\omega}\right).$$

Example 4.3.4 (Reconciliation Attack)**.**
We will now work through an example of the reconciliation attack for small values of UOV parameters. The implementation of the attack with further commentary is available in the appendix (see `ra.sage`).

Fix $n = 5$, $m = 2$, and let the order of the base field be $q = 31$. Despite n and m being so small we have an unbalanced version with $n > 2m$. The first step is to generate a new UOV key $p = f \circ A$. The matrices of the private key are given by

$$A = \begin{bmatrix} 7 & 23 & 6 & 5 & 1 \\ 12 & 29 & 28 & 4 & 8 \\ 8 & 24 & 22 & 5 & 27 \\ 12 & 10 & 14 & 17 & 26 \\ 26 & 28 & 25 & 16 & 9 \end{bmatrix} \quad F_1 = \begin{bmatrix} 19 & 16 & 24 & 7 & 23 \\ 16 & 17 & 14 & 0 & 21 \\ 24 & 14 & 13 & 29 & 18 \\ 7 & 0 & 29 & 0 & 0 \\ 23 & 21 & 18 & 0 & 0 \end{bmatrix} \quad F_2 = \begin{bmatrix} 21 & 21 & 14 & 29 & 14 \\ 21 & 4 & 0 & 30 & 0 \\ 14 & 0 & 25 & 2 & 11 \\ 29 & 30 & 2 & 0 & 0 \\ 14 & 0 & 11 & 0 & 0 \end{bmatrix},$$

and the public key is then

$$P_1 = \begin{bmatrix} 26 & 27 & 8 & 7 & 11 \\ 27 & 27 & 8 & 28 & 18 \\ 8 & 8 & 1 & 9 & 0 \\ 7 & 28 & 9 & 23 & 21 \\ 11 & 18 & 0 & 21 & 26 \end{bmatrix} \quad P_2 = \begin{bmatrix} 1 & 2 & 30 & 14 & 26 \\ 2 & 25 & 27 & 18 & 8 \\ 30 & 27 & 4 & 8 & 8 \\ 14 & 18 & 8 & 15 & 0 \\ 26 & 8 & 8 & 0 & 14 \end{bmatrix}.$$

Since the top-left submatrix $A_1 \in k^{(n-m)\times(n-m)}$ of A is invertible, there is an alternative secret transformation A' of the form (4.3). By Theorem 4.3.1 this transformation can be computed by superimposing $A'_2 = A_1^{-1}A_2$ on top of the identity matrix. This leads to the following equivalent key

$$A' = \begin{bmatrix} 1 & 0 & 0 & 24 & 10 \\ 0 & 1 & 0 & 17 & 27 \\ 0 & 0 & 1 & 11 & 9 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \; F'_1 = \begin{bmatrix} 26 & 27 & 8 & 14 & 4 \\ 27 & 27 & 8 & 11 & 1 \\ 8 & 8 & 1 & 11 & 5 \\ 14 & 11 & 11 & 0 & 0 \\ 4 & 1 & 5 & 0 & 0 \end{bmatrix} \; F'_2 = \begin{bmatrix} 1 & 2 & 30 & 29 & 2 \\ 2 & 25 & 27 & 23 & 0 \\ 30 & 27 & 4 & 25 & 28 \\ 29 & 23 & 25 & 0 & 0 \\ 2 & 0 & 28 & 0 & 0 \end{bmatrix}.$$

We will now recover an alternative private key $(g, B) \in K$ directly from the public key p using reconciliation attack. Let $d = 1$. We setup a system of equations (4.6) in b_* to determine the matrix

$$B_{(1)} = \begin{bmatrix} 1 & 0 & 0 & 0 & b_0 \\ 0 & 1 & 0 & 0 & b_1 \\ 0 & 0 & 1 & 0 & b_2 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

By equating the bottom-right submatrix of $B_{(1)}^{\intercal} P_i B_{(1)}$ to $0_{1\times 1}$ we obtain the following system of equations

$$\begin{cases} 26b_0^2 + 23b_0b_1 + 16b_0b_2 + 22b_0 + 27b_1^2 + 16b_1b_2 + 5b_1 + b_2^2 + 26 = 0 \\ b_0^2 + 4b_0b_1 + 29b_0b_2 + 21b_0 + 25b_1^2 + 23b_1b_2 + 16b_1 + 4b_2^2 + 16b_2 + 14 = 0 \end{cases}.$$

The system of equations is then solved using the methods of Elimination Theory. Since there could be many solutions[4], we pick one and compute the intermediate matrices $P_i^{(1)} =$

[4]The current version of Sage [SD22] does not allow iterating through a variety of dimension greater than zero, so we add linear constraints to reduce the dimension.

$B_{(1)}^{\mathsf{T}} P_i B_{(1)}$, which produces

$$B_{(1)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 20 \\ 0 & 1 & 0 & 0 & 30 \\ 0 & 0 & 1 & 0 & 23 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \; P_1^{(1)} = \begin{bmatrix} 26 & 27 & 8 & 7 & 6 \\ 27 & 27 & 8 & 28 & 2 \\ 8 & 8 & 1 & 9 & 20 \\ 7 & 28 & 9 & 23 & 30 \\ 6 & 2 & 20 & 30 & 0 \end{bmatrix} \; P_2^{(1)} = \begin{bmatrix} 1 & 2 & 30 & 14 & 21 \\ 2 & 25 & 27 & 18 & 24 \\ 30 & 27 & 4 & 8 & 22 \\ 14 & 18 & 8 & 15 & 12 \\ 21 & 24 & 22 & 12 & 0 \end{bmatrix}.$$

There are no guarantees that the chosen solution for $d = 1$ will yield a complete solution for matrix B. On one hand, there are no guarantees that there is matrix B of the expected form (see the probability estimate of Theorem 4.3.1); on the other hand, the chosen solution for $d = 1$ might not be extendable to $d = 2$. Therefore, we have to try all solutions for different values of d. The solution for $d = 1$ above is known to produce a complete solution for B, but unsuccessful candidates were excluded from the example.

We now move to $d = 2$ and setup a system of equations to determine the coefficients of the matrix

$$B_{(2)} = \begin{bmatrix} 1 & 0 & 0 & b_0 & 0 \\ 0 & 1 & 0 & b_1 & 0 \\ 0 & 0 & 1 & b_2 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

For each component of the intermediate public key $P_i^{(1)}$ we now have one linear and one quadratic equation, which matches the statement of Lemma 4.3.2. The equations are as follows

$$\begin{cases} 26b_0^2 + 23b_0b_1 + 16b_0b_2 + 14b_0 + 27b_1^2 + 16b_1b_2 + 25b_1 + b_2^2 + 18b_2 + 23 = 0 \\ 6b_0 + 2b_1 + 20b_2 + 30 = 0 \\ b_0^2 + 4b_0b_1 + 29b_0b_2 + 28b_0 + 25b_1^2 + 23b_1b_2 + 5b_1 + 4b_2^2 + 16b_2 + 15 = 0 \\ 21b_0 + 24b_1 + 22b_2 + 12 = 0 \end{cases}.$$

Upon solving this system of equations we obtain the vector $(b_0, b_1, b_2) = (8, 19, 19)$ which is then substituted into $B_{(2)}$. The alternative private key is given by the matrix $B = (B_{(1)}B_{(2)})^{-1}$ and the components of the central map $G_i = B_{(2)}^{\mathsf{T}} B_{(1)}^{\mathsf{T}} P_i B_{(1)} B_{(2)}$. The recovered key is as follows

$$B = \begin{bmatrix} 1 & 0 & 0 & 23 & 11 \\ 0 & 1 & 0 & 12 & 1 \\ 0 & 0 & 1 & 12 & 8 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \; G_1 = \begin{bmatrix} 26 & 27 & 8 & 12 & 6 \\ 27 & 27 & 8 & 10 & 2 \\ 8 & 8 & 1 & 27 & 20 \\ 12 & 10 & 27 & 0 & 0 \\ 6 & 2 & 20 & 0 & 0 \end{bmatrix} \; G_2 = \begin{bmatrix} 1 & 2 & 30 & 10 & 21 \\ 2 & 25 & 27 & 30 & 24 \\ 30 & 27 & 4 & 0 & 22 \\ 10 & 30 & 0 & 0 & 0 \\ 21 & 24 & 22 & 0 & 0 \end{bmatrix}.$$

4.4 Intersection Attack

The intersection attack of Ward Beullens [Beu21] builds on the Kipnis-Shamir and the reconciliation attacks discussed in the previous sections. We will briefly summarize the attack below.

Definition 4.4.1 (Polar Form)**.** Let $g \in k[w_1, \ldots, w_n]$ be of degree two. The polynomial g' in $k[w_1, \ldots, w_n, z_1, \ldots, z_n]$ is said to be a polar form of g if

$$g'(w, z) = g(w + z) - g(w) - g(z) + g(0),$$

where $w = (w_1, \ldots, w_n)$ and $z = (z_1, \ldots, z_n)$.

This definition naturally extends to m-tuples of polynomials $p = (p_1, \ldots, p_m)$ in R^m

$$p'(w, z) = (p'_1(w, z), \ldots, p'_m(w, z)).$$

Since the elements of R are homogeneous, $p_i(0) = 0$, so the polar forms simplify to $p'_i(w, z) = p_i(w + z) - p(w) - p(z)$. A polar form defines a symmetric and bilinear map [Beu21, Theorem 1], which leads to a number of interesting applications.

Remark 4 (Alternative Characterization of UOV). Recall that if $p \in R^m$ is a UOV public key, then $p(O_x) = 0$ by Lemma 4.2.4. Beullens observed that this property could be used to give an alternative definition of the signature scheme [Beu21, Section 3]. The secret key could be any description of $O_x \subseteq k^n$ and the public key $p \in R^m$ could be any polynomial tuple that vanishes on O_x. To sign a message $b \in k^m$ one fixes $v \in k^n$ and composes a system of equations for $o \in O_x$

$$p(v + o) = p(v) + p(o) + p'(v, o) = b.$$

Since $p(v)$ is a constant, $p(o) = 0$, $p'(v, o)$ is linear in o, and $\dim O_x = m$, we have a system of m linear equations in m variables. If a solution exists, the signature is the vector $v + o \in k^n$.

Intersection Attack Let $p = (p_1, \ldots, p_m) \in R^m$ be a UOV public key and denote by $P_\ell \in k^{n \times n}$ the matrices of quadratic forms corresponding to p_ℓ. If P_i, P_j are invertible, then Lemma 4.2.5 states that

$$\dim (P_i O_x \cap P_j O_x) \geq 3m - n.$$

When $n < 3m$ the intersection of these subspaces is nontrivial. If we find a vector v in this intersection, then $P_i^{-1} v$ and $P_j^{-1} v$ are in the O_x. Having found two vectors in O_x is equivalent to carrying out the reconciliation attack for $d = 1$ and $d = 2$, provided there is a solution of the form (4.3). By Lemma 4.3.2, the remaining steps of the reconciliation attack are easier because we obtain up to $m(d - 1)$ linear constraints at each step. Therefore, the complexity is dominated by finding a vector in the intersection, which gives the name to the attack.

Let $n < 3m$. To find a vector $z = (z_1, \ldots, z_n) \in P_i O_x \cap P_j O_x$ we compose a system of equations in $k[z_1, \ldots, z_n]$ defined by

$$\begin{aligned} p(P_i^{-1} z) &= 0, \\ p(P_j^{-1} z) &= 0, \\ p'(P_i^{-1} z, P_j^{-1} z) &= 0. \end{aligned} \tag{4.7}$$

The first two equalities must hold because $P_i^{-1} z$ and $P_j^{-1} z$ are in O_x and the third one must hold because we also have $P_i^{-1} z + P_j^{-1} z$ in O_x. This yields a system of $3m$ quadratic equations in n variables. We know that the solution space has dimension at least $3m - n$, so we can impose $3m - n$ linear constraints to obtain a system of n equations in n variables. While this is not better than solving the PEP directly, this idea can be generalized.

Let $n < 2.5m$. To optimize the attack we want to increase the number of intersections. For $s > 1$ and $n < \frac{2s-1}{s-1} m$, the author estimates that

$$\dim \left(L_1 O_x \cap \cdots \cap L_s O_x \right) \geq sm - (s-1)(n-m) > 0,$$

where L_i are linear combinations of invertible matrices P_ℓ. This yields a system of equations similar to (4.7) with $M = \binom{s+1}{2} m - 2\binom{s}{2}$ quadratic equations in $N = ns - (2s-1)m$ variables. The complexity of the attack is estimated using Wiedemann XL algorithm (see Section 3.4) to be

$$3 \binom{N + d_{\mathrm{reg}}}{d_{\mathrm{reg}}}^2 \binom{N+2}{2}$$

operations in k. It is precisely because of this attack that the security of parameters of Czypek et al. in Table 2.1 was reduced from 128 to 95 bits.

Chapter 5

Conclusions

A digital signature scheme must prevent signature forgery by parties who do not have access to the private key. This is achieved by employing a computationally hard problem at the foundation of the respective construction. In case of UOV this is the Multivariate Quadratic (MQ) problem, which posits that solving a random system of multivariate quadratic equations is NP-complete [GJ79]. However, using a solid hardness assumption to build a signature scheme does not guarantee the security of the scheme.

There are three security concerns that motivated the study outlined in this work. All of them could render an instantiation of UOV less secure than expected. Informally, the concerns are:

(I) Key space is too small.
(II) Public key leaks information about the central map.
(III) Private key could be recovered by means other than MQ.

The expectation of security is stated in terms of the number of operations in the base field k required to forge a signature for a particular choice of parameters. UOV has three parameters: the number of polynomials m, the number of variables n, and the order q of the field k. Table 2.1 provides several such expectations found in published works. Table 5.1 summarizes complexity of the attacks discussed in this work. Let us review each concern in order.

(I) The key space is studied in Chapter 2. Theorem 2.3.7 provides a complete classification of sustaining transformations of UOV. Counting sustaining transformation identifies redundancies in the key space in the form of equivalent keys. Our result improves on the previous work [WP05] as outlined in Table 2.3. However, even with the increased number of equivalent keys, the key space is still large enough to fulfill the expectations.

(II) The relationships between public and private keys are studied in Section 2.3. We break-

down the $\mathrm{GL}(n, k)$ component of the key space into smaller subsets and illustrate the new components along with the respective relationships in Figure 2.1. While no new attacks have been uncovered, it is perhaps not too surprising. Faugère and Perret have experimentally concluded that UOV public keys behave like semi-regular systems [FP09]. Bulygin, Petzoldt, and Buchmann established a lower bound on the complexity of solving UOV using Gröbner bases [BPB10]. The experiment produced in this work indicates that small UOV instances behave like regular systems (see Table 3.2 and Section 3.2.4). In other words, the UOV public keys appear to hide the structure of the central map reasonably well.

(III) Definition 4.0.1 (PEP) captures the essence of the relationship between the public and private keys of UOV without alluding to the MQ problem. The attack of Kipnis and Shamir (Section 4.2) laid the foundations for solving PEP by showing that elements of O_x can be recovered by considering invariant subspaces of linear maps $P_i^{-1}P_j$. Kipnis, Patarin, and Goubin showed that the intersection $P_i^{-1}P_jO_x \cap O_x$ is nontrivial for $n < 3m$ (Lemma 4.2.5), which motivates the recommendation to set $n \geq 3m$. Reconciliation attack (Section 4.3) provides a general method for solving PEP. While its complexity is not much different from a direct approach, the value of the reconciliation attack is in showing that the complexity of PEP is determined by recovering the first vector in O_x. As of this writing, there appears to be no published solutions to PEP for $n \geq 3m$. This makes UOV one of the few MPKC signature schemes that managed to resist cryptanalysis to date. However, the are also no known reductions of PEP to common complexity classes, which means there could be efficient solutions to this problem. This is a fascinating direction for future research.

Future Research

There are several directions for future work that are worth highlighting:

1. Explore methods of finding vectors in O_x that are not derived from Lemma 4.2.5. By Theorem 4.1.3, a basis of O_x solves PEP. One possible direction is to study the connection between a stabilizer group of a public key $\mathrm{stab}(p)$ and O_x discussed in Sections 2.3 and 4.1 respectively. There is a connection between f and O_x as well as between $\mathrm{stab}(f)$ and $\mathrm{stab}(p)$, but it is unclear if this could be combined into a method for solving PEP. The classification of the UOV key space illustrated in Figure 2.1 is aimed to highlight possible paths.

2. Generalization of the Thomae-Wolf algorithm. This algorithm (Section 3.3) has a potential to provide general means of converting underdetermined systems into determined systems. There are two aspects that limit its applicability:

   (a) Some solutions a_ℓ to the linear systems $L_{i,j}$ produce $\mathbb{V}(g_{j_1}, \ldots, g_{j_{m-t}}) = \emptyset$. If there are no solutions to $g_j = 0$, the most time consuming step has to be re-

Condition	Complexity	Algorithm
$n \geq m$	$O\left(\binom{n+i_{\mathrm{reg}}}{n}^{\omega}\right)$	Solving $p(a) = 0$ using Gröbner bases, provided p is a regular system with $n - m$ linear constraints imposed (Example 3.2.23)
$n > cm$	$O\left(\binom{m-\lfloor c \rfloor+1+i_{\mathrm{reg}}}{n-\lfloor c \rfloor+1}^{\omega}\right)$	Solving $p(a) = 0$ using the Thomae-Wolf algorithm for $c > 2$, provided all partial solutions a_* are found from the first attempt (Section 3.3)
$n \approx \sqrt{m}$	$O\left(m^{\omega}\right)$	Solving $p(a) = b$ using relinearization technique (Lemma 3.4.3)
$n = 2m$	$O\left(n^3\right)$	Executing the Kipnis-Shamir attack, provided B meets the requirements from the first attempt (Section 4.2)
$n > 2m$	$O\left(q^{-(2m-n+1)}n^4\right)$	Kipnis-Patarin-Goubin estimate for solving PEP (Theorem 4.2.6)
$n \geq 2m$	$O\left(\binom{n-m+i_{\mathrm{reg}}}{n-m}^{\omega}\right)$	Executing the reconciliation attack, provided there is a solution B of the required form (Lemma 4.3.3)
$n < 2.5m$	$O\left(\binom{N+d_{\mathrm{reg}}}{d_{\mathrm{reg}}}^2\binom{N+2}{2}\right)$	Executing the intersection attack for $n < \frac{2s-1}{s-1}m$, where $s > 1$ and $N = ns - (2s-1)m$ (Section 4.4)

Table 5.1: Complexity of algorithms discussed in this work. The estimate in the "Complexity" column assumes that all prerequisites outlined in the respective sections are met for the supplied input to the algorithm.

executed, which makes complexity analysis difficult.

(b) Some constraints on m, n, q must be able to yield guaranteed solutions even for fields of odd characteristic. We have identified three places of arbitrary choice in Section 3.3 that could be used for estimating the bounds.

3. Accounting for permutations of the variables. In order for a system of equations to be invertible using the method of Lemma 2.0.4, the system should be linear in m variables x_i. However, the choice of m indices corresponding to the Oil variables can be arbitrary, provided it is consistent for all polynomials in the system. For instance, consider a matrix F of a central map f with respect to parameters $m = 2$, $n = 5$, $q = 31$. Let M_π be a permutation matrix that swaps x_2 with x_4, and let F_π be the composition $F_\pi = M_\pi^\intercal F M_\pi$

$$F = \begin{bmatrix} 16 & 7 & 30 & 7 & 10 \\ 7 & 19 & 17 & 7 & 21 \\ 30 & 17 & 20 & 4 & 25 \\ 7 & 7 & 4 & 0 & 0 \\ 10 & 21 & 25 & 0 & 0 \end{bmatrix} \quad M_\pi = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad F_\pi = \begin{bmatrix} 16 & 7 & 30 & 7 & 10 \\ 7 & 0 & 4 & 7 & 0 \\ 30 & 4 & 20 & 17 & 25 \\ 7 & 7 & 17 & 19 & 21 \\ 10 & 0 & 25 & 21 & 0 \end{bmatrix}.$$

Observe that, by Lemma 2.2.3, F_π is no longer a central map, yet the polynomial f_π, corresponding to F_π, is linear in variables x_2 and x_5. This is easy to see if we fix the Vinegar variables in the respective polynomials

$$f(1, 1, 1, x_4, x_5) = 5x_4 - 12x_5 + 8, \qquad f_\pi(1, x_2, 1, 1, x_5) = 5x_2 - 12x_5 + 8.$$

Furthermore, by Definition 2.2.2, M_π is not a sustaining transformation because it maps F to F_π, which is not a central map. The results of Chapters 2 and 4 could be extended to take into account permutations of the variables:

(a) Study the effect of permutations on further reduction of the key space. Theorem 2.2.4 counts the number of equivalent keys. If $p = f \circ A$ and B is a sustaining transformation, then $p = (f \circ B) \circ (B^{-1}A)$ is easy to solve because $f \circ B$ is a central map. However, the decomposition $p = (f \circ M_\pi) \circ (M_\pi^{-1}A)$ is also easy to solve, yet it is not counted by the results from Chapter 2.

(b) Study if there is any advantage to solving PEP when the definition of a central map also includes matrices such as F_π. By Definition 4.0.1, $C = (M_\pi^{-1}A)^{-1}$ is not a solution to PEP, yet $p \circ C = f \circ M_\pi$ is easy to invert.

References

[AFI+04] Gwénolé Ars, Jean-Charles Faugère, Hideki Imai, Mitsuru Kawazoe, and Makoto Sugita. Comparison between XL and Gröbner basis algorithms. In *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*, volume 3329 of *Lecture Notes in Computer Science*, pages 338–353. Springer, 2004.

[AW20] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication, 2020. https://arxiv.org/abs/2010.05846.

[Ber67] E. R. Berlekamp. Factoring polynomials over finite fields. *The Bell System Technical Journal*, 46(8):1853–1859, 1967.

[Beu21] Ward Beullens. Improved cryptanalysis of UOV and Rainbow. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, pages 348–373, Cham, 2021. Springer International Publishing.

[Beu22] Ward Beullens. Breaking Rainbow takes a weekend on a laptop. Cryptology ePrint Archive, Paper 2022/214, 2022. https://eprint.iacr.org/2022/214.

[BFS03] Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. Complexity of Gröbner basis computation for semi-regular overdetermined sequences over $\mathbb{F}_2$ with solutions in $\mathbb{F}_2$. Research Report RR-5049, INRIA, 2003. Available at https://hal.inria.fr/inria-00071534.

[BFS04] Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. On the complexity of Gröbner basis computation of semi-regular overdetermined algebraic equations. In *International Conference on Polynomial System Solving*, pages 71–75, 2004. Available at https://www-polsys.lip6.fr/~jcf/Papers/43BF.pdf.

[BFS13] Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. On the complexity of the F_5 Grörbner basis algorithm, 2013. https://arxiv.org/abs/1312.1655.

[BPB10] Stanislav Bulygin, Albrecht Petzoldt, and Johannes Buchmann. Towards provable security of the Unbalanced Oil and Vinegar signature scheme under direct attacks. In Guang Gong and Kishan Chand Gupta, editors, *Progress in Cryptology - INDOCRYPT 2010*, pages 17–32, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[BPSV19] Ward Beullens, Bart Preneel, Alan Szepieniec, and Frederik Vercauteren. LUOV signature scheme proposal for NIST PQC project (Round 2 version). NIST Post-Quantum Cryptography Standardization Process, Second Round Submissions, 2019. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-2-submissions.

[BW93] Thomas Becker and Volker Weispfenning. *Gröbner Bases*. Graduate Texts in Mathematics. Springer New York, NY, first edition, 1993.

[CHT12] Peter Czypek, Stefan Heyse, and Enrico Thomae. Efficient implementations of MQPKS on constrained devices. In *CHES*, volume 7428, pages 374–389. Springer, 2012.

[CKPS00] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 392–407. Springer, 2000.

[CLO15] David A. Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms*. Springer International Publishing, fourth edition, 2015.

[DDVY21] Jintai Ding, Joshua Deaton, Vishakha, and Bo-Yin Yang. The nested subset differential attack: A practical direct attack against LUOV which forges a signature within 210 minutes. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, pages 329–347, Cham, 2021. Springer International Publishing.

[DPS20] Jintai Ding, Albrecht Petzoldt, and Dieter S. Schmidt. *Multivariate Public Key Cryptosystems*. Springer US, 2020.

[DS05] Jintai Ding and Dieter Schmidt. Rainbow, a new multivariable polynomial signature scheme. In John Ioannidis, Angelos Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security*, pages 164–175, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[DYC+08] Jintai Ding, Bo-Yin Yang, Chia-Hsin Owen Chen, Ming-Shing Chen, and Chen-Mou Cheng. New differential-algebraic attacks and reparametrization of Rainbow. In Steven M. Bellovin, Rosario Gennaro, Angelos Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security*, pages 242–257, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[Fau02] Jean Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5). In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, ISSAC '02, pages 75–83, New York, NY, USA, 2002. Association for Computing Machinery.

[FP09] Jean-Charles Faugère and Ludovic Perret. On the security of UOV. Cryptology ePrint Archive, Paper 2009/483, 2009. `https://eprint.iacr.org/2009/483`.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Mathematical Sciences Series. W. H. Freeman, 1979.

[Her75] Israel Nathan Herstein. *Topics in Algebra*. John Wiley & Sons, second edition, 1975.

[Hum96] John F. Humphreys. *A Course in Group Theory*. Oxford Graduate Texts in Mathematics. Oxford University Press, 1996.

[KG85] Walter Keller-Gehrig. Fast algorithms for the characteristic polynomial. *Theoretical Computer Science*, 36(2–3):309–317, June 1985.

[KPG99] Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced Oil and Vinegar signature schemes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, pages 206–222, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. Extended version avaialable at `http://goubin.fr/papers/OILLONG.PDF`.

[KS98] Aviad Kipnis and Adi Shamir. Cryptanalysis of the Oil and Vinegar signature scheme. In Hugo Krawczyk, editor, *Advances in Cryptology — CRYPTO '98*, pages 257–266, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[KS99] Aviad Kipnis and Adi Shamir. Cryptanalysis of the HFE public key cryptosystem by relinearization. In *Advances in Cryptology - CRYPTO '99, 19th Annual Inter-*

national Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings, volume 1666 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 1999.

[Laz83] D. Lazard. Gröbner bases, Gaussian elimination and resolution of systems of algebraic equations. In J. A. van Hulzen, editor, *Computer Algebra*, pages 146–156, Berlin, Heidelberg, 1983. Springer Berlin Heidelberg.

[Mac02] F. S. Macaulay. Some formulae in elimination. *Proceedings of the London Mathematical Society*, s1-35(1):3–27, 1902.

[Moh11] Wael Said Abdelmageed Mohamed. *Improvements for the XL Algorithm with Applications to Algebraic Cryptanalysis*. PhD thesis, Technische Universität, Darmstadt, June 2011.

[Nat22] National Institute of Standards and Technology. Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process. Technical Report NISTIR 8413, U.S. Department of Commerce, Washington, D.C., July 2022.

[Pat96] Jacques Patarin. Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): Two New Families of Asymmetric Algorithms. In *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, volume 1070 of *Lecture Notes in Computer Science*, pages 33–48. Springer, 1996.

[Pat97] Jacques Patarin. The Oil and Vinegar signature scheme. In *Dagstuhl Workshop on Cryptography September, 1997*, 1997.

[SD22] W. A. Stein and The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.5)*, 2022. `https://www.sagemath.org`.

[Ser73] Jean-Pierre Serre. *A Course in Arithmetic*. Graduate Texts in Mathematics. Springer New York, NY, 1973.

[Str69] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

[TW10] Enrico Thomae and Christopher Wolf. Solving systems of multivariate quadratic equations over finite fields or: from relinearization to MutantXL. Cryptology ePrint Archive, Paper 2010/596, 2010. `https://eprint.iacr.org/2010/596`.

[TW12] Enrico Thomae and Christopher Wolf. Solving underdetermined systems of multivariate quadratic equations revisited. In *Public Key Cryptography - PKC 2012*,

volume 7293 of *Lecture Notes in Computer Science*, pages 156–171. Springer, 2012.

[WP05] Christopher Wolf and Bart Preneel. Large superfluous keys in multivariate quadratic asymmetric systems. In *Public Key Cryptography - PKC 2005, 8th International Workshop on Theory and Practice in Public Key Cryptography, Les Diablerets, Switzerland, January 23-26, 2005, Proceedings*, volume 3386 of *Lecture Notes in Computer Science*, pages 275–287. Springer, 2005.

Appendix A

SageMath Programs

The programs below are written in SageMath [SD22] using a dialect of the Python programming language. To run the experiments:

1. Save the respective files into a directory on your computer,
2. Open a new session in SageMath,
3. Load the program of interest.

For example, to run the reconciliation attack experiment, save `uov.sage` and `ra.sage` into a directory, then run the following from a command line:

```
$ cd path/to/programs
$ sage
sage: load("ra.sage")
sage: # paste examples from ra.sage
```

A.1 Unbalanced Oil and Vinegar

The following program implements the UOV key generation, signing, and signature verification. Since `uov.sage` implements common procedures used by other programs this file must be available on disk for examples to work.

uov.sage

```
# Key generation and common procedures for Unbalanced Oil and Vinegar.
reset()

# generate_keys returns a random UOV instance (P, F, T) for the specified
# field k, number of variables n, and number of polynomials m.
def generate_keys(k, n, m, order="deglex"):
        R = PolynomialRing(k, ["x%d" % i for i in range(1,n+1)], order=order)
        M = MatrixSpace(k, n)
```

```
        # Generate central maps.
        F = []
        for i in range(m):
                x = R.gens()
                f = 0
                for i in range(n - m):
                        start_index = i
                        for j in range(start_index, n):
                                f += k.random_element() * x[i] * x[j]
                F.append(f)

        # Generate the secret transformation T.
        T = M.random_element()
        while not T.is_invertible():
                T = M.random_element()

        # Compose the central maps with T.
        P = [T.act_on_polynomial(f) for f in F]

        return P, F, T

# is_central_map returns true if all elements of the specified list of
# matrices Q have zero submatrix corresponding to the oil variables.
def is_central_map(Q, m):
        for M in Q:
                n, _ = M.dimensions()
                if not M.submatrix(n-m, n-m).is_zero():
                        return False
        return True

# poly_to_matrix takes a homogeneous polynomial f of degree two and returns a
# matrix corresponding to the quadratic form f. When characteristic of k is
# odd, the returned matrix is symmetric, provided the flag is set to True.
def poly_to_matrix(f, symmetric=True):
        assert f.is_homogeneous() and f.degree() == 2, "f is not homogeneous"

        R = f.parent()
        k = R.base_ring()
        n = len(R.gens())

        rows = []
        for i in range(n):
                row = [0] * n
                for j in range(i, n):
                        m = R.gen(i) * R.gen(j)
                        c = f.monomial_coefficient(m)
                        row[j] = c
                rows.append(row)

        Q = matrix(k, rows)
        if symmetric and k.characteristic() != 2:
                Q = (Q + Q.transpose()) / 2

        # The symmetric matrix for fields of characteristic 2 is defined by
        # Q = Q + Q.transpose(), but the operations of poly_to_matrix and
```

```
        # matrix_to_poly become incompatible. So we only return symmetric
        # matrices for fields of odd characteristic for now.
        return Q

# matrix_to_poly returns the polynomial corresponding to the quadratic form
# Q, where variables are formed by the generators of the polynomial ring R.
def matrix_to_poly(R, Q):
        x = vector(R.gens())
        return x * Q * x

# compose_qf returns a composition of the quadratic form Q with the linear
# transformation A.
def compose_qf(Q, A):
        if "matrix" in dir(A):
                A = A.matrix()
        return A.transpose() * Q * A

# subs replaces the variables of f with the components of the vector v. f can
# either be a polynomial or a matrix.
def subs(f, v):
        s = {}
        gens = f.parent().gens()
        if "submatrix" in dir(f):
                gens = f.base_ring().gens()
        for i in range(len(gens)):
                s[gens[i]] = v[i]
        return f.subs(s)

# polar_form returns the value of the polar form at (x, y).
def polar_form(f, x, y):
        return subs(f, x + y) - subs(f, x) - subs(f, y)

# find_solutions solves the multivariate quadratic system of equations
# specified by ideal I. All solutions are returned when dim I = 0. If
# dim I = 1 random linear constraints are added to reduce the dimension and
# the operation is retried. An empty list is returned when dim I > 1 or dim I
# < 0 (no solutions).
def find_solutions(I):
        dim = I.dimension()
        if dim < 0 or dim > 1:
                # print("find_solutions with dim(I) = %d; returning []" % dim)
                return []
        elif dim == 0:
                return I.variety()

        # dimension 1
        sols = []
        R = I.gen(0).parent()
        x_i = R.gen(randint(0, len(R.gens())-1))
        for val in I.base_ring():
                J = ideal(I.gens() + [x_i + val])
                sols += find_solutions(J)

        return sols

# invert_central_map returns a pre-image value a such that F(a) = b.
```

```
def invert_central_map(F_polys, b, debug=False):
        R_x = F_polys[0].parent()
        k = R_x.base_ring()
        n = len(R_x.gens())
        m = len(F_polys)

        # It should be possible to invert the central map in q^(n-m) attempts.
        for _ in range(k.order()^(n-m)):
                a = []
                fixed_vars = {}
                for i in range(n-m):
                        v = k.random_element()
                        fixed_vars[R_x.gen(i)] = v
                        a.append(v)
                if debug: print("fixed_vars =", fixed_vars)

                F_prime = [f.subs(fixed_vars) for f in F_polys]
                if debug: print("F_prime =", F_prime)

                M = []
                for f in F_prime:
                        row = []
                        for i in range(n-m, n):
                                row.append(f.monomial_coefficient(R_x.gen(i)))
                        M.append(row)

                M = matrix(M)
                if debug:
                        print("M =")
                        pretty_print(M)

                if M.is_invertible():
                        b_prime = []
                        for i in range(len(F_prime)):
                                v = F_prime[i].constant_coefficient()
                                b_prime.append(b[i] - v)

                        b_prime = vector(b_prime)
                        if debug: print("b_prime =", b_prime)

                        M_inv = M.inverse()
                        sol = M_inv * b_prime
                        a.extend(sol)

                        return vector(a)

        # This may happen for small values of q, m, n.
        assert false, "failed to invert the central map (try a different key)"

# verify returns true if P(a) == b for all components of the public key P.
def verify(P, a, b):
        b_prime = vector([subs(p, a) for p in P])
        return b_prime == b

# sign returns a signature vector a such that P(a) = b, where P is the
# public key (P = F o T).
```

```
def sign(F, T, b):
        a = invert_central_map(F, b)
        return T.inverse() * a

# oilspace_y returns O_y = span(e_{n-m}, ..., e_n).
def oilspace_y(k, n, m):
        basis = []
        for i in range(m):
                v = [0 for j in range(n)]
                v[n - i - 1] = 1
                basis.append(v)

        return span(k, basis)

# oilspace_x returns O_x, which is a pre-image of O_y with respect to T.
def oilspace_x(n, m, T):
        k = T.base_ring()
        T_inv = T.inverse()

        basis = []
        for b in oilspace_y(k, n, m).basis():
                basis.append(T_inv * b)

        return span(k, basis)

# test_central_map ensures that the central map vanishes on O_y.
def test_central_map(F, num_tests=10):
        R = F[0].parent()
        k = R.base_ring()
        n = len(R.gens())
        m = len(F)
        O_y = oilspace_y(k, n, m)

        for i in range(num_tests):
                o = O_y.random_element()
                subs = {}
                for i in range(len(R.gens())):
                        subs[R.gen(i)] = o[i]

                for i in range(len(F)):
                        # print("F_%d(%s) = %s" % (i, subs, F[i].subs(subs)))
                        assert F[i].subs(subs) == 0

# test_public_key ensures that the public key vanishes on the O_x.
def test_public_key(P, O_x, num_tests=10):
        R = P[0].parent()
        k = R.base_ring()

        for i in range(num_tests):
                v = O_x.random_element()
                subs = {}
                for i in range(len(v)):
                        subs[R.gen(i)] = v[i]
                for i in range(len(v), len(R.gens())):
                        subs[R.gen(i)] = 0
```

```
                for i in range(len(P)):
                        # print("P_%d(%s) = %s" % (i, subs, P[i].subs(subs)))
                        assert P[i].subs(subs) == 0

# test_poly_matrix verifies that the matrix to poly conversion works as
# expected.
def test_poly_matrix(polys):
        R = polys[0].parent()
        k = R.base_ring()
        n = len(R.gens())

        G = GL(n, k)
        for f in polys:
                # test poly/matrix conversions
                R = f.parent()
                Q = poly_to_matrix(f)
                g = matrix_to_poly(R, Q)

                assert f == g, "poly_matrix 1"

                # test group actions
                A = G.random_element()
                f = A.matrix().act_on_polynomial(f)
                Q = compose_qf(Q, A)
                g = matrix_to_poly(R, Q)

                assert f == g, "poly_matrix 2"

# define and execute a test suite for the UOV programs.
def run_tests():
        n = 5
        m = 2
        for q in [2, 3, 31]:
                k = GF(q, "z")
                P, F, T = generate_keys(k, n, m)

                O_x = oilspace_x(n, m, T)
                test_central_map(F)
                test_public_key(P, O_x)
                test_poly_matrix(F + P)
                Q = [poly_to_matrix(f) for f in F]
                assert is_central_map(Q, m)

                b = vector(k, [k.random_element() for i in range(m)])
                a = sign(F, T, b)
                assert verify(P, a, b), "sign/verify failed"

run_tests()

# sign_verify_example generates a new set of keys, signs a message, and
# verifies the signature. It prints all the intermediate results.
def sign_verify_example():
        # Generate a new key pair
        P_polys, F_polys, A = generate_keys(GF(31), 5, 2)
        P = [poly_to_matrix(p, symmetric=False) for p in P_polys]
        F = [poly_to_matrix(f, symmetric=False) for f in F_polys]
```

```
        print("Central map")
        pretty_print(F)
        print("Public key")
        pretty_print(P)
        print("Secret linear transformation")
        pretty_print(A)

        # Invert the central map.
        mu = [7, 14]
        sigma_prime = invert_central_map(F_polys, mu, True)

        # Confirm the central map inverted correctly.
        print("Message =", mu)
        print("Pre-image of the central map =", sigma_prime)
        print("Evaluating F_polys at sigma_prime")
        for i in range(len(F)):
                res = sigma_prime * F[i] * sigma_prime
                print("  F_%d%s = %d" % (i, sigma_prime, res))

        # Invert the public key and confirm the inversion is successful.
        sigma = A.inverse()*sigma_prime
        print("Pre-image of the public key =", sigma)
        print("Evaluating P_polys at sigma")
        for i in range(len(P)):
                res = sigma * P[i] * sigma
                print("  P_%d%s = %d" % (i, sigma, res))

# permutation_example illustrates that a different choice of oil variables
# produces a polynomial with the properties similar to UOV central maps.
def pemutation_example():
        # Generate a random central map.
        k = GF(31)
        _, F_polys, _ = generate_keys(k, 5, 2)

        f = F_polys[0]
        F = poly_to_matrix(f)
        R = F_polys[0].parent()

        print("A randomly generated central map is given by")
        print("f =", f)
        print("The matrix form of f is F =")
        pretty_print(F)

        # The oil variables from Definition 2.0.3 are x_4 and x_5. We swap x_4
        # with x_2 using the following permutation matrix.
        M_pi = matrix(k, [[1, 0, 0, 0, 0],
                          [0, 0, 0, 1, 0],
                          [0, 0, 1, 0, 0],
                          [0, 1, 0, 0, 0],
                          [0, 0, 0, 0, 1]])

        # F_pi is no longer a central map as per Definition 2.0.3.
        F_pi = compose_qf(F, M_pi)
        print("Matrix of the central map F with x_4 and x_2 swapped is F_pi =")
        pretty_print(F_pi)
```

```
        # However, F_pi is linear in x_2, x_4, which can be inverted using
        # the same procedure as in Lemma 2.0.4.
        f_pi = matrix_to_poly(R, F_pi)
        print("The polynomial corresponding to F_pi is")
        print("f_pi =", f_pi)
        V = R.gens_dict()
        print("f_pi(1, x_2, 1, 1, x_5) =",
                f_pi.subs({V["x1"]: 1, V["x3"]: 1, V["x4"]: 1}))
        print("f   (1, 1, 1, x_4, x_5) =",
                f.subs({V["x1"]: 1, V["x2"]: 1, V["x3"]: 1}))

# Example 1: generate keys, sign a message, verify the signature.
#
# sign_verify_example()

# Example 2: different choice of oil variables x_{n-m+1}, ..., x_n.
#
# pemutation_example()
```

A.2 Thomae-Wolf Algorithm

The program below implements the Thomae-Wolf algorithm as described in Section 3.3.

tw.sage

```
# Thomae-Wolf algorithm for solving MQ systems for m = c*n with c > 2.
reset()
load("uov.sage")

# For enumerating monomial degrees.
from itertools import combinations_with_replacement

def poly_to_vector(f, b):
        row = []
        for g in f.parent().gens():
                row.append(f.monomial_coefficient(g))
        b.append(-f.constant_coefficient())
        return row

# decompose finds a matrix B such that B^T P B = F, where F is a matrix
# suitable for Thomae and Wolf algorithm.
def decompose(R_x, P):
        m = len(P)
        n, _ = P[0].dimensions()
        R_b = PolynomialRing(k, n-1, "b", order="lex")

        B = identity_matrix(k, n)
        for d in range(1, m):
                B_d = identity_matrix(R_b, n)
                col = list(R_b.gens())
                col.insert(d, 1)
                B_d.set_column(d, col)
                print("B_d")
```

```
                pretty_print(B_d)

                M = []
                b = []
                eqs_per_p = min(t, d)
                eqs = []
                for p in P:
                        P_d = compose_qf(p, B_d)
                        for i in range(eqs_per_p):
                                eq = P_d[i, d]
                                eqs.append(eq)
                                # print("P_d[%d, %d] = %s" % (i, d, eq))
                                M.append(poly_to_vector(eq, b))

                M = matrix(M)
                b = vector(b)
                print("M =")
                pretty_print(M)
                print("b =")
                pretty_print(b)

                sol = M.solve_right(b)
                ker = M.right_kernel()
                sol = sol + ker.random_element()

                assert M*sol == b, "wrong solution B_d (1)"
                for eq in eqs:
                        assert subs(eq, sol) == 0, "wrong solution B_d (2)"
                print("sol =", sol)

                B_d = subs(B_d, sol)
                print("B_(%d)" % d)
                B = B*B_d
                pretty_print(B)

                P_d = [compose_qf(p, B_d).change_ring(k) for p in P]
                print("P_(%d)" % d)
                pretty_print(P_d)

                P = P_d
        return P, B

# linear_eqs returns the system of linear equations L_{i,j} in a matrix form.
def linear_eqs(F):
        m = len(F)
        n, _ = F[0].dimensions()
        t = floor(n/m) - 1
        L = []

        for f in F:
                for i in range(t):
                        eq = f.row(i).list()
                        eq = eq[m:n]
                        L.append(eq)

        return matrix(L)
```

```
# macaulay_matrix converts the specified list of polynomials to the Macaulay
# matrix using monomial order of the underlying polynomial ring.
def macaulay_matrix(polys):
        R_x = polys[0].parent()

        M = []
        for g in polys:
                assert g.degree() == 2, "g is not of degree 2"

                row = []
                for d in [2, 1]:
                        for c in combinations_with_replacement(R_x.gens(), d):
                                m = prod(c)
                                row.append(g.monomial_coefficient(m))
                row.append(g.constant_coefficient())
                M.append(row)
        return matrix(M)

# macaulay_matrix_to_polys returns a list of polynomials corresponding to rows
# of the specified Macaulay matrix M using the monomial order and generators
# from the polynomial ring R_x.
def macaulay_matrix_to_polys(R_x, M):
        num_eqs, _ = M.dimensions()
        eqs = []
        for i in range(num_eqs):
                j = 0
                g = 0
                for d in [2, 1]:
                        for c in combinations_with_replacement(R_x.gens(), d):
                                m = prod(c)
                                g += M[i, j]*m
                                j += 1
                # constant term
                g += M[i, j]
                eqs.append(g)
        return eqs

# Example execution of the algorithm.
k = GF(order=31, name="z", repr="int")
n = 17
m = 5
t = floor(n/m) - 1
P_polys, central_map_polys, A = generate_keys(k, n, m)

# P = [poly_to_matrix(p, symmetric=False) for p in P_polys]
P = [poly_to_matrix(p) for p in P_polys]

print("Public key P")
pretty_print(P)
print("Central map F")
pretty_print([poly_to_matrix(f) for f in central_map_polys])
print("Secret transformation A")
pretty_print(A)

R_x = P_polys[0].parent()
```

```
R_y = PolynomialRing(k, m-t, "y", order="lex")

F, B = decompose(R_x, P)
for i in range(len(P)):
        assert compose_qf(P[i], B) == F[i], "decomposition of P"

# if m divides n, we might get an invertible L, then there's only a trivial
# solution.
L = linear_eqs(F)
print("Linear equations L")
pretty_print(L)

sol = L.right_kernel().random_element()
a_linear = {}
for i in range(m, n):
        a_linear[R_x.gen(i)] = sol[i-m]
print("Using the following solution to L x = 0: ", sol)

F_polys = [matrix_to_poly(R_x, g) for g in F]
print("Polynomials F")
pretty_print(F_polys)

F_polys = [g.subs(a_linear) for g in F_polys]
print("Polynomials F after substituting a solution to L")
pretty_print(F_polys)

print("Matrices of homogeneous components of F after substitution")
pretty_print([poly_to_matrix(g.homogeneous_components()[2]) for g in F_polys])

# Randomly select t polynomials from F
idx = [0 .. (len(F_polys)-1)]
shuffle(idx)
G_polys = [F_polys[idx[i]] for i in range(t)]

M = macaulay_matrix(G_polys)
M = M.echelon_form()
G_polys = macaulay_matrix_to_polys(R_x, M)
print("Polynomials G in reduced row echelon form")
pretty_print(G_polys)

# Prepare the substitutions x_i^2 = Q(...)
quad_subs = {}
for i in range(t):
        mon = R_x.gen(i)^2
        g = G_polys[i]
        assert g.monomial_coefficient(mon) == 1, ("invalid g[%d]" % i)
        quad_subs[mon] = mon - g

print("Quadratic substitutions")
pretty_print(quad_subs)

# Substitute into remaining F_polys to obtain the system of equations in y
var_subs = {}
inv_var_subs = {}
for i in range(t, m):
        x = R_x.gen(i)
```

```
        y = R_y.gen(i-t)
        var_subs[x] = y
        inv_var_subs[y] = x

I = []
for i in range(t, len(F_polys)):
        g = F_polys[idx[i]]
        for mon in quad_subs:
                c = g.monomial_coefficient(mon)
                print("g   =", g)
                if c != 0:
                        g -= c*mon
                        print("g'  =", g)
                        g += c*quad_subs[mon]
                        print("g'' =", g)

        # Map g to a polynomial in y
        g = R_y(g.subs(var_subs))
        I.append(g)

print("System I of (m-t) eqs in (m-t) variables")
pretty_print(I)
I = ideal(I)
a_g_sols_in_y = I.variety()
print("V(I) =", a_g_sols_in_y)

a_complete = {}
for a_g_y in a_g_sols_in_y:
        # Map a candidate solution of V(I) in y_i vars to x_i vars
        a_g = {}
        for y in a_g_y:
                a_g[inv_var_subs[y]] = a_g_y[y]

        a_sqrt = {}
        for mon in quad_subs:
                v = quad_subs[mon].subs(a_g)
                if is_square(v):
                        a_sqrt[sqrt(mon)] = sqrt(v)

        if len(a_sqrt) == len(quad_subs):
                print("a_sqrt =", a_sqrt)
                # The complete solution a_complete is a concatenation of
                #   1. Square roots a_sqrt      x_0,     ..., x_t
                #   2. Solution to V(I) a_g     x_{t+1}, ..., x_m
                #   3. Linear solution a_linear x_{m+1}, ..., x_n
                for x in a_sqrt:
                        a_complete[x] = a_sqrt[x]
                for x in a_g:
                        a_complete[x] = a_g[x]
                for x in a_linear:
                        a_complete[x] = a_linear[x]
                break

if len(a_complete) != n:
        # Square roots might not exist for partial solutions, so it is likely
        # we will end up here when working over fields of odd characteristic.
```

```
        print("No solutions found")
else:
        print("Solution to F(a) = 0")
        pretty_print(a_complete)

        print("Evaluating F(a)")
        for i in range(len(F_polys)):
                print("  F[%d](a) = %s" % (i, F_polys[i].subs(a_complete)))

        print("Solution to P(a') = 0")
        sol = []
        for i in range(n):
                sol.append(a_complete[R_x.gen(i)])
        sol = vector(k, sol)
        B = matrix(k, B)
        sol = B*sol
        pretty_print(sol)

        print("Evaluating P(a')")
        for i in range(len(P_polys)):
                print("  P[%d](a') = %s" % (i, subs(P_polys[i], sol)))
```

A.3 Reconciliation Attack

The program below implements the reconciliation attack described in Section 4.3. It uses the same notation as the discussion above.

ra.sage

```
# Reconciliation attack on UOV.
reset()
load("uov.sage")

# equations_for_B returns an ideal containing the equations in b_ij variables
# of matrix B_d. The input parameters are:
#
#  R_b - polynomial ring in b_ij variables
#  d   - current iteration 1..m
#  P   - matrices P[0],...,P[m] of quadratic forms from the previous step d-1
#
def equations_for_B(n, m, R_b, d, P):
        B_2 = zero_matrix(R_b, n-m, m)
        B_2.set_column(m-d, R_b.gens())

        B_2_T = B_2.transpose()

        eqs = []
        for e in range(len(P)):
                P_1 = P[e][0:n-m, 0:n-m]
                P_2 = P[e][0:n-m, n-m:n]
                P_3 = P[e][n-m:n, 0:n-m]
                P_4 = P[e][n-m:n, n-m:n]
```

```
                BPB_4 = B_2_T*P_1*B_2 + B_2_T*P_2 + P_3*B_2 + P_4

                # Since P is symmetric, we only need to read row m-d.
                for j in range(m-d, m):
                        eqs.append(BPB_4[m-d,j])

        return R_b.ideal(eqs)

# reconciliation_attack searches for an alternative secret linear
# transformation B for the specified public key P.
def reconciliation_attack(R_x, R_b, P, d):
        k = R_x.base_ring()
        n = len(R_x.gens())
        m = len(P)

        # The attack works for UOV by iterating over the solution space
        # of equations for B. This iteration is inefficient for n > 2m
        # and large values of n and m due to the number of solutions to
        # consider.
        I = equations_for_B(n, m, R_b, d, P)
        V = find_solutions(I)
        print("  d = %d found %d solutions" % (d, len(V)))

        for sol in V:
                B_d = identity_matrix(k, n)
                for i in range(len(sol)):
                        B_d[i, n-d] = sol[R_b.gen(i)]

                P_d = [B_d.transpose() * p * B_d for p in P]

                # Stop the recursion.
                if len(P) == d and is_central_map(P_d, m):
                        return P_d, [B_d], [I]

                # Proceed with recursion.
                P_m, B_m, I_m = reconciliation_attack(R_x, R_b, P_d, d+1)
                if len(P_m) != 0 and len(B_m) != 0:
                        return P_m, B_m + [B_d], I_m + [I]

        return [], [], []

def test_reconciliation_attack(q, n, m):
        k = GF(order=q, name="z", repr="int")
        P_polys, F_polys, A = generate_keys(k, n, m)

        P = [poly_to_matrix(p) for p in P_polys]
        F = [poly_to_matrix(f) for f in F_polys]
        print("Public key P")
        pretty_print(P)
        print("Central map F")
        pretty_print(F)
        print("Secret transformation A")
        pretty_print(A)

        R_x = P_polys[0].parent()
        R_b = PolynomialRing(k, n-m, "b", order="lex")
```

```
        A_1 = A[0:n-m, 0:n-m]
        A_prime = None
        if not A_1.is_invertible():
                print("Warning: A_1 is not invertible")
        else:
                A_2_prime = A_1.inverse() * A[0:n-m, n-m:n]
                A_prime = matrix.identity(k, n)
                A_prime[0:n-m, n-m:n] = A_2_prime
                print("Alternative secret transformation A' (computed from A)")
                pretty_print(A_prime)
                print("Alternative central map F'")
                pretty_print([compose_qf(p, A_prime.inverse()) for p in P])

        print("Executing the reconciliation attack ...")
        G, B_ds, I = reconciliation_attack(R_x, R_b, P, 1)
        G_polys = [matrix_to_poly(R_x, g) for g in G]
        B = mul(B_ds).inverse()

        print("Recovered central map G")
        pretty_print(G)
        print("Recovered secret transformation B")
        pretty_print(B)
        print("B_ds =")
        pretty_print(B_ds)
        print("Equations for B")
        for J in I:
                pretty_print(J.gens())

        # Test that the recovered private key enables signature forgery.
        b = vector(k, [k.random_element() for i in range(m)])
        print("Test message b =", b)
        a = sign(G_polys, B, b)
        print("Forged signature a =", a)
        assert verify(P_polys, a, b), "signature forgery failed"

        return P, F, A, A_prime, G, B_ds, I

P, F, A, A_prime, G, B_ds, I = test_reconciliation_attack(31, 5, 2)
```

A.4 UOV Statistics

The following program computes statistics presented in Section 2.4 and Section 3.2.4.

stats.sage

```
# UOV statistics.
reset()
load("uov.sage")
from numpy import mean

# Tuples (m, n, q).
uov_params = [
        (2, 5, 2),
```

```
        (2, 5, 31),
        (44, 103, 2^8), # Czypek 2012
        (48, 144, 2^8), # Ding   2020
        (64, 192, 2),   # Wolf   2005
]

# Tuples (m, n). Parameters are chosen to be easily computable on a relatively
# modern computer.
toy_params = [
        (2, 4),  (2, 6),
        (3, 6),  (3, 9),
        (5, 10), (5, 12), (5, 14), (5, 15),
]

# Counts of the respective components of the UOV key space.
def num_R(m, n, q):
        return q^((1/2)*n*(n+1))

def num_S(m, n, q):
        return q^((1/2)*(n-m)*(n+m+1))

def num_GL(n, q):
        return prod([(q^n - q^i) for i in range(n)])

# Number of equivalent keys using Wolf 2005 estimate.
def wolf_reduction(m, n, q):
        p1 = prod([(q^m - q^i) for i in range(m)])
        p2 = prod([(q^(n-m) - q^i) for i in range(n-m)])
        return p1*p2

# Number of equivalent keys using Theorem 2.2.4.
def theorem224_reduction(m, n, q):
        p1 = prod([(q^n - q^(n-m+i)) for i in range(m)])
        p2 = prod([(q^(n-m) - q^i) for i in range(n-m)])
        return p1*p2

# Statistics provided in Table 2.2.
def key_space_count(params):
        fmt = "%-4s%-4s%-4s%-12s%-12s%-12s%-12s"
        print(fmt % ("m", "n", "q", "log_2 |R^m|", "log_2 |S^m|",
                "log_2 |GL|", "log_2 |K|"))
        for m, n, q in params:
                Rm = num_R(m, n, q)^m
                Sm = num_S(m, n, q)^m
                GL = num_GL(n, q)

                log_Rm = int(log(Rm, 2))
                log_Sm = int(log(Sm, 2))
                log_GL = int(log(GL, 2))
                log_K = int(log(Sm*GL, 2))
                print(fmt % (m, n, q, log_Rm, log_Sm, log_GL, log_K))
        print()

# Statistics provided in Table 2.3.
def key_space_with_reductions(params):
        fmt = "%-4s%-4s%-4s%-12s%-16s%-12s%-16s%-12s"
```

```
        print(fmt % ("m", "n", "q", "log_2 |K|", "log_2 (K' red)", "log_2 |K'|",
                "log_2 (K'' red)", "log_2 |K''|"))
        for m, n, q in params:
                Rm = num_R(m, n, q)^m
                Sm = num_S(m, n, q)^m
                GL = num_GL(n, q)
                K = Sm * GL
                wolf_red = wolf_reduction(m, n, q)
                t224_red = theorem224_reduction(m, n, q)

                log_K = int(log(K, 2))
                log_wolf_red = int(log(wolf_red, 2))
                log_t224_red = int(log(t224_red, 2))
                log_wolf_K = int(log(K / wolf_red, 2))
                log_t224_K = int(log(K / t224_red, 2))
                print(fmt % (m, n, q, log_K, log_wolf_red, log_wolf_K,
                        log_t224_red, log_t224_K))
        print()

# Statistics provided in Table 2.4.
def prob_count(params):
        fmt = "%-4s%-4s%-4s%-12s%-12s"
        print(fmt % ("m", "n", "q", "P[g in S]", "P[f o A in S]"))

        for m, n, q in params:
                S = num_S(m, n, q)
                R = num_R(m, n, q)
                P_g_in_S = q^((-1/2)*m*(m+1))
                assert P_g_in_S == S/R, "S/R"

                P_fA_in_S = prod(
                        [(q^(n-m) - q^i)/(q^n - q^i) for i in range(n-m)]
                        )

                log_P_g_in_S = int(log(P_g_in_S, 2))
                log_P_fA_in_S = int(log(P_fA_in_S, 2))
                print(fmt % (m, n, q, log_P_g_in_S, log_P_fA_in_S))
        print()

# Compute the index of regularity ireg by comparing the value of the Hilbert
# polynomial at i with the respective value of the Hilbert function. We use the
# Hilbert series to obtain the values of the Hilbert function and consider the
# first ~20 terms (this value is determined by the precision of HS as returned
# by HS.prec()).
def index_of_regularity(I):
        PSR = PowerSeriesRing(QQ, "z")
        HP = I.hilbert_polynomial(algorithm="singular")
        HS = PSR(I.hilbert_series())
        # print("I =", I)
        # print("Hilbert series =", HS)
        # print("Hilbert polynomial =", HP)

        ireg = 0
        HF = HS.list() # returns HS.prec() terms in a list
        for i in range(len(HF)):
                if HF[i] != HP(i):
```

```
                        ireg = i + 1

        return ireg

# Statistics provided in Table 3.1.
def dim_dmax_stats(params, q, num_tests):
        fmt = "%-4s%-4s%-12s%-12s%-12s%-12s%-12s%-12s"
        print("Using q=%d and num_tests=%d for the table below"
                % (q, num_tests))
        print(fmt % ("m", "n", "dim_min", "dim_avg", "dim_max",
                "dmax_min", "dmax_avg", "dmax_max"))

        for m, n in params:
                dim_vals = []
                dmax_vals = []
                for _ in range(num_tests):
                        P_polys, F_polys, A = generate_keys(GF(q), n, m, "degrevlex")
                        I = ideal(P_polys)
                        G = I.groebner_basis()
                        I = ideal(G)
                        dim = I.dimension()
                        dmax = max([g.degree() for g in G])
                        dim_vals.append(I.dimension())
                        dmax_vals.append(dmax)

                print(fmt % (m, n, min(dim_vals), "%.2f" % mean(dim_vals),
                        max(dim_vals), min(dmax_vals), "%.2f" % mean(dmax_vals),
                        max(dmax_vals)))

# gen_random_constr returns "count" homogeneous linear constraints over R. The
# constraints are linearly independent.
def gen_random_constr(R, count):
        k = R.base_ring()
        n = len(R.gens())
        c = GL(n, k).random_element().matrix()

        constr = []
        for i in range(count):
                # c_1 x_1 + ... + c_n x_n = 0
                eq = sum([c[i][j]*R.gen(j) for j in range(n)])
                constr.append(eq)
        return constr

def _constr(m, n, k):
        P_polys, _, _ = generate_keys(k, n, m, "degrevlex")

        R = P_polys[0].parent()
        I = ideal(P_polys)
        dim_I = I.dimension()

        # Impose dim V(I) random constraints
        constr = gen_random_constr(R, dim_I)

        J = ideal(P_polys + constr)
        dim_J = J.dimension()
        ireg_J = index_of_regularity(J)
```

```
        dmax_J = max([g.degree() for g in J.groebner_basis()])
        hs_poly = 0
        if J.hilbert_series() in QQ["t"]:
                hs_poly = 1
        dmax_est = sum([g.degree() - 1 for g in J.gens()]) + 1

        return dim_J, ireg_J, dmax_J, dmax_est, hs_poly

# Statistics provided in Table 3.2.
def stats_with_constr(params, q, num_tests):
        fmt = "%-4s%-4s%-6s%-16s%-16s%-12s%-16s"
        print("Using q=%d and num_tests=%d for the table below"
                % (q, num_tests))
        print(fmt % ("m", "n", "D_max", "dmax_J != D_max", "ireg_J != D_max",
                "dim_J != 0", "HS_J is poly"))

        k = GF(q)
        for m, n in params:
                dim_J_vals = []
                ireg_J_vals = []
                dmax_J_vals = []
                dmax_est_vals = []
                hs_poly_vals = []
                for _ in range(num_tests):
                        dim, ireg, dmax, dmax_est, hs_poly = _constr(m, n, k)
                        dim_J_vals.append(dim)
                        ireg_J_vals.append(ireg)
                        dmax_J_vals.append(dmax)
                        dmax_est_vals.append(dmax_est)
                        hs_poly_vals.append(hs_poly)

                assert min(dmax_est_vals) == max(dmax_est_vals), "unexpected D_max"
                D_max = dmax_est_vals[0]

                print(fmt % (m, n,
                        D_max,
                        len(dmax_J_vals) - dmax_J_vals.count(D_max),
                        len(ireg_J_vals) - ireg_J_vals.count(D_max),
                        len(dim_J_vals) - dim_J_vals.count(0),
                        sum(hs_poly_vals),
                        ))

# Example 3.2.23.
def example_uov_complexity():
        P_polys, _, _ = generate_keys(GF(31), 5, 2)
        print("Public key P")
        pretty_print([poly_to_matrix(p) for p in P_polys])

        I = ideal(P_polys)
        dim_I = I.dimension()
        print("dim V(I) =", dim_I)

        R = P_polys[0].parent()
        constr = gen_random_constr(R, dim_I)
        print("Random constraints")
        pretty_print(constr)
```

```
        J = ideal(P_polys + constr)
        dim_J = J.dimension()
        print("dim V(J) =", dim_J)
        print("    V(J) =", J.variety())

        HS = J.hilbert_series()
        HP = J.hilbert_polynomial()
        print("HS_J =", HS)
        print("HP_J =", HP)
        print("ireg =", index_of_regularity(J))
        print("dmax est =", sum([g.degree() - 1 for g in J.gens()]) + 1)
        print("dmax act =", max([g.degree() for g in J.groebner_basis()]))

# Example 1: complexity of solving UOV.
#
# example_uov_complexity()

# Statistics provided in tables of Chapter 2.
#
# key_space_count(uov_params)
# key_space_with_reductions(uov_params)
# prob_count(uov_params)

# Statistics provided in tables of Chapter 3.
#
# dim_dmax_stats(toy_params, 31, 100)
# stats_with_constr(toy_params, 31, 100)
```